Benemérita Universidad Autónoma De Puebla

Facultad De Ciencias De La Computación

Una propuesta de modelado cognitivo en animación por computadora

Tesis que para obtener el título de:

Licenciado En Ingeniería En Tecnologías De La Información

Presenta: Jorge Alberto López Aragón

Asesores: Dr. Abraham Sánchez López

Mc. Juan Carlos Conde Ramírez

Septiembre de 2021

Agradecimientos

A mi madre, quien me ha acompañado a lo largo de toda mi vida, brindándome la motivación necesaria para seguir adelante, quien con sus enseñanzas me permitió crecer como persona convirtiéndome en el hombre que soy hoy en día. Le agradezco por brindarme su apoyo cuando más lo necesite, gracias por permitirme cumplir mis sueños.

A mi asesor y amigo el Dr. Abraham Sánchez López, por su dedicación y esfuerzo en formar buenos profesionistas, tuve la fortuna de que tenerlo como mi maestro y mentor a lo largo de mi carrera, gracias por poner a prueba mi potencial haciéndome crecer profesionalmente.

A mi co-asesor y amigo el MC. Juan Carlos Conde Ramírez, le agradezco su paciencia y su apoyo en el desarrollo del presente trabajo, porque gracias a sus clases aumento mi interés en el campo de los videojuegos, permitiéndome desarrollar el presente trabajo.

A mis mejores amigos Iván y Jesús, así como a mis grandes amigos de mi vida universitaria Laura, Esteban, Jiuber y Juan Pablo, gracias por su apoyo y por su confianza plena en mí, por animarme a seguir siempre adelante, por las vivencias que pasamos juntos, ustedes siempre serán una parte importante de mi vida, la familia que yo elegí.

Índice General

Índice de Figuras

Capítulo 1

1 Introducción

A lo largo de los años los videojuegos han hecho uso de técnicas de inteligencia artificial cada vez más complejas para proporcionar a los usuarios un entorno con una experiencia inmersiva cada vez mayor. Para ello, en algunos casos es preciso hacer uso de personajes autónomos que se ocupen de cumplir operaciones determinadas, por lo tanto, es necesario el uso de métodos que permitan generar comportamientos “realistas” para dichos personajes. Con el paso del tiempo los videojuegos se hicieron cada vez más complejos, con ello, se fueron desarrollando nuevas técnicas viables que permitieron generar un comportamiento cada vez más realista o natural tanto del ambiente como de los personajes. En este primer capítulo se muestra la transcendencia del estudio del modelado cognitivo y su importancia en el desarrollo de aplicaciones “inteligentes”, es decir, aplicaciones capaces de modelar personajes que no solo tengan un comportamiento geométrico clásico, sino que muestren aspectos más realistas que permiten obtener un comportamiento más natural mediante el modelado cognitivo.

1.1. Planteamiento del problema

Para dotar a un personaje de comportamiento inteligente es necesario afrontar varios retos, entre ellos la dificultad de proporcionarle instrucciones claras sobre las acciones que debe realizar, esto es, porque el agente por sí mismo desconoce la naturaleza de dichas instrucciones, de manera inicial este no cuenta con un modelo de percepción de su propio mundo, por ello es esencial dotar al agente de aspectos importantes tales como percepción, razonamiento y juicio [1]. Para ello es imprescindible el uso de modelos y técnicas que permitan al agente abstraer la información de manera adecuada, permitiéndole así el interactuar con su ambiente.

La elección de un modelo en específico depende en gran medida del nivel de cognición con el cual se deseé que cuente el agente, algunos modelos pueden proporcionar resultados más óptimos que otros, sin embargo, esto depende en gran medida del problema que se quiera resolver, es decir, el tipo de entorno y escenario con el cual se verá involucrado nuestro agente.

1.2. Antecedentes

Los primeros modelos computacionales utilizados por animadores y diseñadores de videojuegos fueron los modelos geométricos, estos permiten capturar la apariencia y forma de un personaje, hacen uso de la cinemática directa e inversa la cual permite aplicar restricciones en las partes de un modelo que se encuentran vinculadas entre sí, pese a ser los primeros modelos con ellos es posible controlar de manera muy efectiva personajes en animaciones y videojuegos. Los modelos físicos permiten la automatización de la animación de objetos pasivos, por ejemplo, objetos que caen, objetos en colisión, partículas, objetos deformables, etc., actualmente es común que los modelos físicos se incorporen en paquetes de animación o en motores de videojuegos. Los modelos relacionados con la biomecánica permiten simular la locomoción de seres vivos plasmándola en personajes y criaturas virtuales como peces, serpientes, aves, etc. El modelado de comportamiento permite que un agente pueda reaccionar a estímulos de su ambiente. El modelado cognitivo surge del intento de automatizar aún más el proceso de generación de comportamientos. Estos son el siguiente paso lógico en la

jerarquía de los modelos utilizados en la animación y videojuegos (Figura 1.1). Estos permiten elevar el nivel de abstracción en los personajes dando resultado a comportamientos de alto nivel [2].

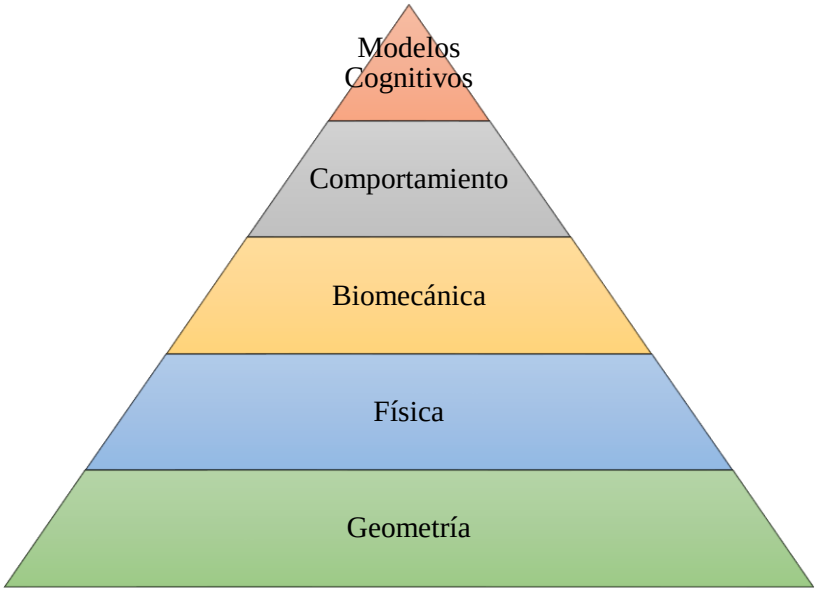


Figura 1.1. Jerarquía de modelos usados en videojuegos y animación.

1.3. Modelado Cognitivo

El modelado cognitivo es un campo creciente de la computación cognitiva el cual tiene como propósito identificar que procesos y estrategias se emplean para efectuar tareas de nivel superior [3], permitiendo así el desarrollo de modelos de procesos cognitivos.

En el campo de la ciencia cognitiva una de las herramientas más utilizadas son los modelos cognitivos. En medicina, los neuropsicólogos hacen uso de ellos para evaluar las diferencias de comportamiento entre personas sanas y pacientes con alguna patología como, por ejemplo, la esquizofrenia. En neurociencia cognitiva, sirven para comprender la función psicológica de las diferentes áreas del cerebro. También se utilizan para entender el proceso de envejecimiento y el deterioro de las funciones cognitivas con la edad [4].

En [5] John Funge menciona que el modelado cognitivo para juegos y animación explora la interfaz provocadora, pero en gran parte inexplorada entre los gráficos por computadora y la inteligencia artificial. En la animación y videojuegos los modelos cognitivos no se limitan

únicamente a los modelos de comportamiento, ya que estos rigen lo que sabe un personaje, el cómo adquiere dicho conocimiento y como puede utilizarlo para planificar sus futuras acciones.

El modelado cognitivo al encontrarse en el nivel superior de la jerarquía de los modelos utilizados en la animación y videojuegos (Figura 1.1.) funciona en conjunto con los niveles inferiores permitiendo así, brindar una experiencia visualmente atractiva.

1.4. Objetivo

El objetivo principal de esta investigación es proponer algunas estrategias que permitan implementar los aspectos cognitivos en la animación por computadora, elevando con ello el nivel de abstracción para que el usuario pueda definir el comportamiento de un personaje de forma directa. Por lo tanto, se plantean los siguientes objetivos específicos:

- Revisar las últimas propuestas del modelado cognitivo que se han publicado en los últimos años por parte de la comunidad de graficas por computadora.
- Desarrollar un ejemplo de animación por computadora que contenga aspectos de cognición (facultad de procesar a partir de la percepción, experiencia y características subjetivas que permitan valorar cierta información).
- Evaluar esta propuesta con el trabajo realizado como parte inicial de la tesis, una animación simple sin modelado cognitivo. Realizar un estudio comparativo.

Capítulo 2

2 Estado del Arte

La computación cognitiva está generando un gran cambio en la forma en cómo se interactúa con las aplicaciones, ya que involucra diferentes actividades tales como aprendizaje, interpretación, razonamiento, adaptación, entre algunas otras. Es por ello que permite tener aplicaciones con capacidades para entender el lenguaje humano, comprender textos e imágenes, así como de aprender y brindar respuestas con un nivel de confianza alto.

En el campo de la animación y los videojuegos, la computación cognitiva proporcionar un nivel de abstracción a los personajes y entidades. El presente capitulo describe los avances más sobresalientes en los últimos años para generar herramientas de modelado cognitivo y personajes autónomos.

2.1 Personajes Cognitivos

Un personaje 3D posee un modelo propio interno de su mundo virtual, es decir, tiene una percepción propia de su entorno, dicho modelo interno es conocido como modelo cognitivo.

Los modelos cognitivos se aplican para el control de personajes altamente autónomos que se encuentran en la producción de animación y videojuegos.

Es común asumir que el comportamiento de un personaje se encuentra determinado por la secuencia de acciones que ejecuta, no obstante, esta es una simplificación sencilla debido a que es posible que el resultado de dichas acciones pueda depender de factores externos que se encuentren más allá del control del personaje. Por ejemplo, en un videojuego donde se tienen enemigos, el objetivo principal de estos será dañar al personaje principal, si los enemigos atacan al jugador con un arma y estos la pierden deben encontrar otra forma con la cual hacer daño al jugador, generando así un comportamiento diferente.

Existen personajes autónomos, que no poseen conocimiento del dominio en el que se encuentran, por ello, es necesario para ellos decidir de antemano las acciones que deben realizar abarcando todas las diferentes situaciones en las que podrían encontrarse, este tipo de comportamiento se define como determinista. En los videojuegos los jugadores se aburren rápidamente de la naturaleza previsible de los NPC (*Non Playable Character*) que hacen uso de un comportamiento determinista. Por lo tanto, para brindar una mejor experiencia es necesario el uso de personajes cuyos comportamientos no necesiten estar completamente determinados de antemano, es decir, se requiere de personajes que puedan elegir su propio comportamiento en función de la tarea o dirección que se les dé, este tipo de comportamiento se define como no determinista [2]. Es importante aclarar que un comportamiento no determinista no es lo mismo que un comportamiento aleatorio. Por el contrario, el personaje debe ser capaz de elegir sus secuencias de acción de una manera específica para la tarea que debe realizar. Algunas de las secuencias de acción tendrán efectos deseables y otras tendrán efectos indeseables. Es posible indicarle al personaje que efectos son deseables estableciendo objetivos, con ello podrá buscar secuencias de acción que considere le permitan alcanzarlos.

En [2] John Funge menciona que cuando un personaje selecciona acciones de forma no determinista basándose en una meta que está tratando de lograr, nos referimos a su comportamiento posterior como dirigido a una meta.

Particularmente el comportamiento determinista es rápido de ejecutar, ya que el programador ha definido todas las posibles acciones por adelantado, en cambio, el comportamiento dirigido a una meta conlleva una mayor carga de trabajo ya que el personaje tiene que calcular y planificar sus posibles acciones en tiempo de ejecución.

Mayormente una aplicación hará uso de más de un solo enfoque para poder generar el comportamiento de un personaje, esto se debe a que el uso de diferentes enfoques permite dar solución a una mayor cantidad de problemas. En particular, a menudo tiene sentido implementar comportamientos simples de "bajo nivel" como comportamientos deterministas predefinidos, reservando especificaciones dirigidas a objetivos para el comportamiento de "alto nivel".

2.2 Adquisición de Conocimientos

En [2] se menciona que para un personaje el tipo de conocimiento más simple que puede adquirir por sí mismo es la información sobre el estado actual de su mundo, este tipo de adquisición de conocimiento es comúnmente conocido como *sensación*. El otro tipo de conocimiento que un personaje puede necesitar es el conocimiento del dominio sobre las dinámicas de su mundo, este tipo de adquisición de conocimiento se denomina *aprendizaje*.

Es necesario tener presente que se debe tener cuidado en cómo se representa el conocimiento del dominio de un personaje, ya que en función de cómo se representa afectara en gran medida el razonamiento de un personaje sobre su secuencia de acciones.

Para que un personaje pueda adquirir conocimiento es necesario que exista un modelo computacional del mundo virtual, este consiste en un conjunto de reglas y estados que se aplican al estado actual del mundo y que permiten obtener un estado nuevo, este es conocido como el modelo del mundo real.

No es recomendable el proporcionar el modelo del mundo real a un personaje para que conozca por completo su entorno, ya que esto puede generar algunos problemas, los cuales se listan a continuación:

- Eficiencia: De manera normal un personaje deberá contar con una serie de cursos de acción alternativos, el considerar cada secuencia de acción posible ubicada en el modelo del mundo real requerirá una simulación complicada, ya que en cada nuevo ciclo se intentará generar el cálculo de todos los cursos se acción alternativos una y otra vez, causado que el proceso de toma de decisiones sea increíblemente lento.

- Comportamiento poco realista: Los personajes con acceso al modelo del mundo real pasan a ser clarividentes, esta capacidad de ver el futuro de sus mundos puede resultar en un comportamiento con apariencia antinatural.

Generalmente, habrá más de un personaje en el mundo. Incluso si todos los personajes son de la misma clase, es normal que lo que cada uno de ellos "conozca" sobre el mundo pueda ser muy diferente. Es decir, cada personaje es autónomo e independiente, con ello, se asegura un nivel de realismo normalmente requerido en animaciones y juegos. Además, se simplifica la tarea de instruirlos, ya que solo es necesario ocuparse de uno de ellos a la vez.

2.3 Inteligencia Artificial

La inteligencia es un rasgo particular de los seres vivos, gracias a la cual les es posible adaptarse a su medio ambiente mediante la comprensión de su entorno, así como de sus características. La inteligencia no se limita únicamente a la capacidad de pensar, sino que también, es la capacidad de un ser vivo de poder percibir y entender el mundo su alrededor, utilizándolo para alcanzar sus objetivos.

En [6] se nos menciona que la inteligencia artificial consiste en dotar a las maquinas con el don del pensamiento racional, para que con ello puedan explotar el medio ambiente en el que se encuentren. Esta se halla presente en gran parte de la historia de los videojuegos. Una de sus primeras aplicaciones fueron los agentes inteligentes capaces de jugar por su cuenta tanto contra otros sistemas de inteligencia artificial como contra humanos. Su objetivo es por ende el tomar las mejores decisiones para ganar el juego, para ello perciben el entorno, recopilan datos y toman la acción más conveniente para ellos.

Con la creciente popularidad de los videojuegos, en la década de los 70’s hubo una llamada época de oro de los videojuegos en la cual la inteligencia artificial comenzó a centrarse en crear enemigos convincentes para los videojuegos de un solo jugador. Con el paso de los años el uso de sistemas de inteligencia artificial se hizo más popular y común en los videojuegos, con ello algunas aplicaciones interesantes comenzaron a aparecer, por ejemplo, el enfoque inicial de Pac-Man, en el cual los enemigos tenían personalidades distintas y trataban de

perseguir al jugador combinando esfuerzos, usando estrategias reales basadas en técnicas de búsqueda de caminos y de aplicación de patrones para perseguir al jugador.

Tras la popularidad de las aplicaciones basadas en inteligencia artificial, en la edad de oro los videojuegos comenzaron a implementar comportamientos inteligentes con una mayor complejidad. En particular, en la década de 1990 los videojuegos comenzaron a implementar técnicas formales de inteligencia artificial, algunas de las cuales se presentan a continuación:

2.3.1 Búsqueda en Anchura

El algoritmo de búsqueda en anchura (BFS) es uno de los algoritmos de búsqueda en grafos más importantes sobre el que se basan algoritmos más complejos y ampliamente utilizados, este garantiza que, en caso de existir, encontrara la ruta más corta que conecta dos nodos dados en un grafo.

En el algoritmo de búsqueda en anchura se comienza desde un nodo especifico, lo primero que se realiza es verificar que no es el nodo que se está buscando, finalmente se revisan todos sus nodos adyacentes. Si no se encuentra el nodo deseado en los nodos adyacentes se verifican los nodos adyacentes de los nodos que se acaban de verificar y así sucesivamente. Para recrear la ruta que conecta el punto de partida con el punto de meta, simplemente se retrocede desde el nodo meta a través de los nodos vecinos reconstruyendo la ruta más corta. Es importante resaltar que el agregar una propiedad de profundidad a los nodos nos permitirá decidir cuál de los vecinos es parte del camino más corto.

El método de búsqueda en anchura toma como parámetros los nodos inicial y final y devuelve una lista de nodos que es la ruta final que conecta el nodo inicial con el nodo objetivo.

Para implementarlo es necesario realizar un seguimiento de todos los nodos que queremos visitar y todos los nodos que ya visitamos para que no se revise un mismo nodo dos veces. Para ello se hace uso de dos estructuras de datos, una cola y una lista [6]. El funcionamiento de la búsqueda en anchura se muestra en la Figura 2.1.

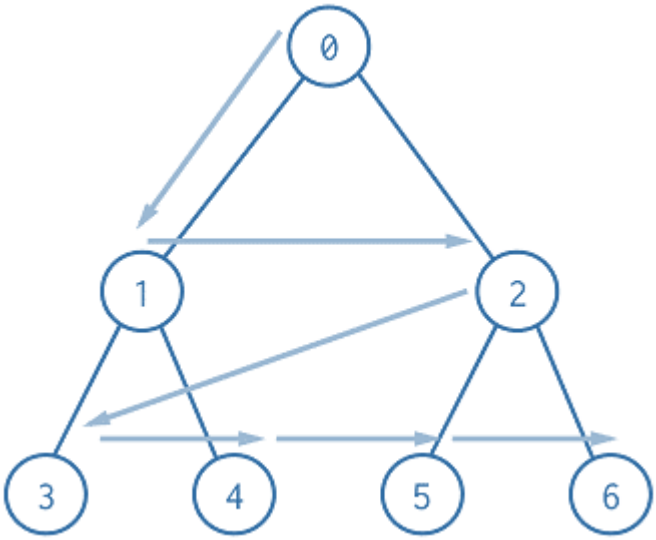


Figura 2.1. Funcionamiento de la búsqueda en anchura (BFS)

2.3.2 Búsqueda en Profundidad

Otro algoritmo muy importante es la búsqueda en profundidad (DFS). Como se explicó en la anterior subsección la búsqueda en anchura busca el nodo observando todas las posibles ramas en paralelo, el algoritmo de búsqueda en profundidad elige un camino y baja hasta el final. Si no se encuentra la meta, retrocede y prueba con otra rama no visitada.

El algoritmo de búsqueda en profundidad puede ser muy útil cuando tiene un árbol muy profundo; en este caso, este algoritmo puede encontrar la solución antes que el algoritmo de búsqueda en anchura. La búsqueda en anchura funciona mejor con árboles muy anchos y no muy profundos, pero la principal diferencia es que la búsqueda en anchura siempre encuentra la ruta más corta, mientras que la búsqueda en profundidad solo encuentra una ruta, no necesariamente la más corta [6].

La búsqueda en profundidad se implementa con frecuencia como un algoritmo recursivo, pero dependiendo del tamaño del árbol y los recursos disponibles, también se puede implementar como un algoritmo iterativo. El funcionamiento de la búsqueda en profundidad se muestra en la Figura 2.2.

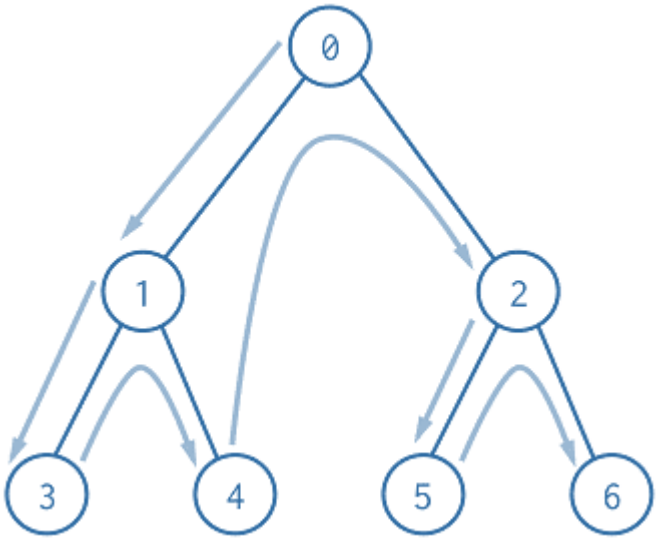


Figura 2.2. Funcionamiento de la búsqueda en profundidad (DFS)

2.3.3 Algoritmo A*

Un algoritmo de búsqueda de tipo ‘primero el mejor’ es un algoritmo de búsqueda que explora un grafo dando prioridad a los nodos más prometedores. La utilidad de expandir o no un nodo se mide mediante una función heurística que permite al algoritmo organizar los nodos en una cola de prioridad, que sugiere el orden en el que deben expandirse.

Una heurística en la resolución de problemas es un enfoque práctico que no promete ser optimo, pero es lo suficientemente rápido y bueno para lograr una meta a corto plazo o para encontrar una solución satisfactoria a un problema.

En el caso de los algoritmos de búsqueda de tipo ‘primero el mejor’ una heurística tiene el propósito de encontrar una estrategia de expansión de nodos que probablemente conduzca a la ruta más corta mediante la evaluación de los nodos a medida que aparecen.

El algoritmo más famoso de este tipo es A* (A-star), y también es el estándar de facto para resolver problemas complejos de búsqueda de caminos (especialmente en espacios 3D) en videojuegos.

El algoritmo A* asigna a cada nodo una puntuación F = G + H, donde:

- G es el costo del camino para llegar al nodo actual desde el inicio.

- H es la heurística del nodo, indica la estimación del costo de un camino optimo desde el nodo actual hasta un estado final.
- F es la estimación del costo total de una solución óptima que pasa por el nodo actual.

Cada vez que el agente A* expande un nodo, asigna una puntuación F a todos los nodos circundantes y pasa al que tiene la puntuación más baja.

Este método permite al agente llegar al objetivo rápidamente sin sufrir el efecto secundario de la búsqueda en anchura de verse obligado a expandir todos los nodos del grafo.

Está claro que A* proporciona un rendimiento general muy superior. También es importante señalar que la eficiencia y, en particular, la complejidad temporal de A* dependen en gran medida de la heurística utilizada. Una buena heurística le dará un buen grado de eficiencia, mientras que una mala podría invalidad el algoritmo por completo [6]. Este algoritmo sigue siendo muy popular en la implementación de los videojuegos actuales, a tal grado que algunos motores de videojuegos como Unity lo incorporan por defecto.

2.3.4 Máquinas de Estado Finito

La máquina de estado finito (FSM) puede resolver elegantemente un amplio conjunto de problemas que afectan a casi todos los géneros de videojuegos, aunque probablemente las aplicaciones más populares son los comportamientos de los enemigos en juegos de acción; en este tipo de juegos los enemigos tienen que reaccionar ante situaciones y ejecutar acciones apropiadas, esos comportamientos pueden representarse como una máquina de estado finito [6].

El uso de la máquina de estado finito no es cosa del pasado, de hecho, podemos encontrar su implementación en juegos modernos y complejos como la saga de Tomb Raider, o las sagas de Call of Duty y Battlefield. Generalmente la mayoría de los juegos están implementando máquina de estado finito para gestionar los componentes de sus NPC. Esto se debe a que son fáciles de implementar y tienen un bajo impacto en el rendimiento. Otros enfoques de inteligencia artificial requieren muchos más recursos y trabajo, y en algunos géneros y escenarios de juegos no recrean un resultado más inmersivo en comparación con su

equivalente en máquina de estado finito. Un ejemplo del diseño de una máquina de estado finito se muestra en la Figura 2.3.

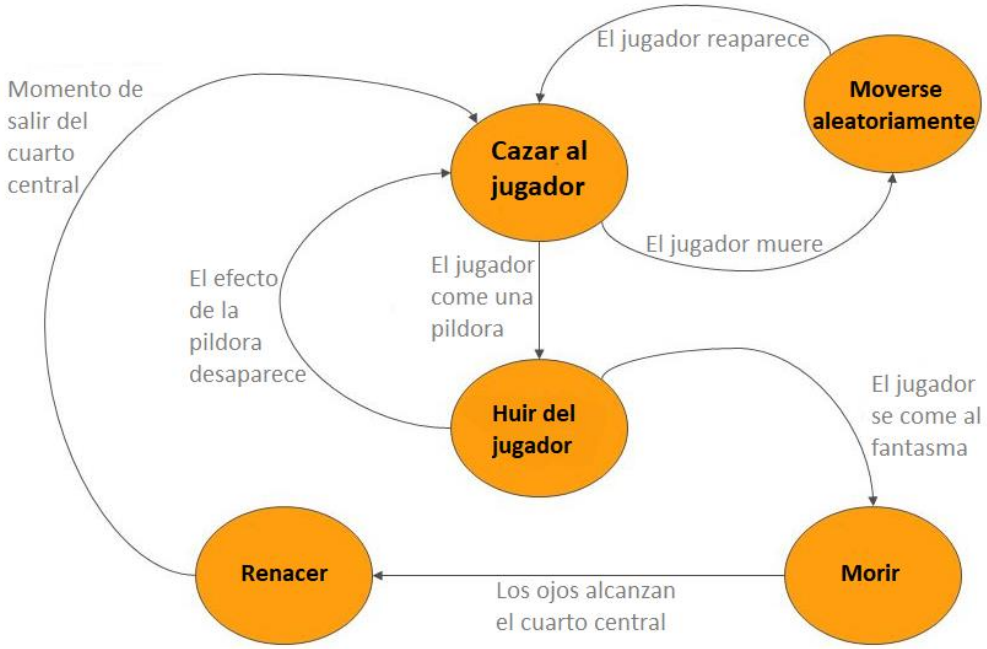


Figura 2.3. Máquina de estado finito diseñada para los fantasmas de Pac-Man.

Capítulo 3

3 Enfoque basado en mapas cognitivos difusos

En este capítulo, se presentan las características y funcionamiento de un sistema de comportamiento cognitivo, así como los detalles de su desarrollo e implementación enfocado en un ambiente virtual submarino que contiene peces y tiburones.

3.1 Mundos Virtuales Difusos

Según Dickerson y Kosko [7] un mundo virtual une a los humanos y las computadoras en un medio causal que puede engañar a la mente o los sentidos. En un sentido más general, un mundo virtual es un sistema dinámico, el cual cambia con el tiempo a medida que el usuario o un personaje se mueve a través de él. En general ambos, tanto el usuario como el mundo virtual deben tener la capacidad de cambiarse el uno al otro.

Cuando un cambio sucede en un mundo virtual este es causal, es decir, los personajes pueden causar eventos conforme se mueven a través de su entorno, generando así nuevos patrones de causa y efecto sobre los ya establecidos. A su vez, el mundo virtual influye en los personajes ya sea en su comportamiento físico, o entorno social, generando de igual maneta un cambio en la red de causa y efecto del personaje. Esta retroalimentación causal entre los personajes y su entorno virtual conforma un sistema dinámico complejo que permite modelar eventos, actores, acciones y datos a medida que se estos desarrollan en el tiempo.

Los mundos virtuales son difusos, así como su retroalimentación. Los eventos ocurren y los conceptos se mantienen solo hasta cierto punto, es decir, los eventos se causan unos a otros hasta cierto punto.

3.2 Mapas Cognitivos Difusos

Los mapas cognitivos difusos (FCM) pueden estructurar mundos virtuales que cambian con el tiempo, estos permiten vincular eventos causales, actores, valores, metas y tendencias en un sistema dinámico de retroalimentación difusa. Un mapa cognitivo difuso enumera las reglas difusas o las rutas de flujo causales que relacionan eventos. Puede guiar a los personajes en un mundo virtual a medida permitiéndoles moverse a través de una red de causa y efecto, con ello, estos pueden reaccionar a los eventos de su entorno, así como a otros personajes. Si un FCM es complejo dará como resultado un mundo virtual con un comportamiento caótico, en cambio, los FCMs simples dan como resultado mundos virtuales con comportamiento periódico [7]. En particular para el presente trabajo se hizo uso de un FCM simple pero adaptable para moldear el entorno submarino.

3.2.1 Funcionamiento

Los mapas cognitivos difusos permiten modelar el mundo virtual en fragmentos. Modelan la red causal como un grafo dirigido difuso donde los nodos y las aristas muestran como los conceptos causales se afectan entre si hasta cierto punto en el sistema dinámico difuso. Los nodos del grafo pueden representar conceptos como eventos, acciones, valores, estados de

ánimo, metas o tendencias, mientras que las aristas establecen reglas difusas o flujos causales entre conceptos. Por ejemplo, el concepto causal **amenaza de supervivencia** permite medir el grado en que ocurre dicho evento difuso. En el mundo submarino donde habitan depredadores y presas, la amenaza de supervivencia aumenta la intención de huir por parte de las presas. La intención de huir aumenta o disminuye en función del grado de amenaza. La regla difusa establece cuánto crece o decrece un nodo a medida que otro crece o decrece.

Los mapas cognitivos difusos pueden estructurar mundos virtuales, por ejemplo, en la figura 3.1 un tiburón encuentra un banco de peces, el tiburón ataca y los peces huyen. Los sistemas difusos integrados controlan sistemas difusos de nivel inferior para la animación, el sonido y otras salidas del mundo virtual [7].

Es común que los mapas cognitivos difusos se representen como grafos causales los cuales no establecen ecuaciones, sino que, declaran nodos conceptuales y los vinculan a otros nodos. El mapa cognitivo difuso convierte cada grafo en una matriz de pesos de reglas difusas. El sistema pondera y agrega las matrices para combinar cualquier número de acciones causales.

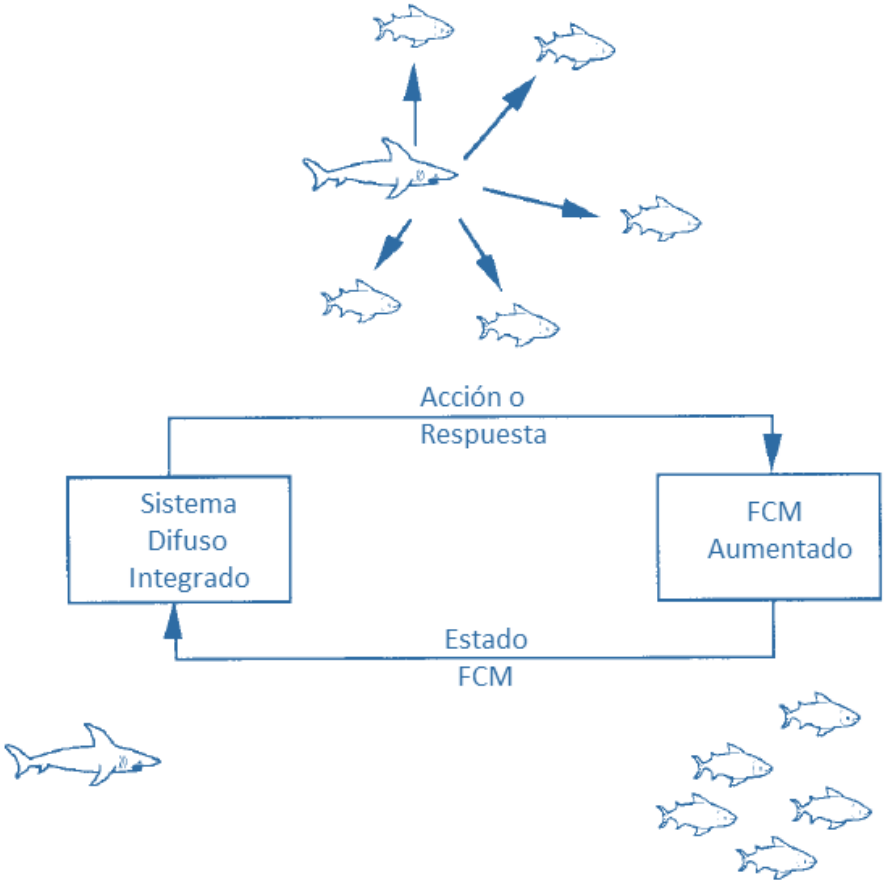


Figura 3.1. Los mapas cognitivos difusos pueden estructurar mundos virtuales.

Los mapas cognitivos difusos se representan como dígrafos con signos de retroalimentación. Los nodos representan conjuntos difusos o eventos que ocurren hasta cierto punto. Los bordes dirigidos representan reglas difusas o el flujo causal parcial entre los conceptos. El signo (+ o -) de un borde significa el aumento o la disminución de su causalidad.

La regla del margen positivo establece que una amenaza de supervivencia aumenta la intención de huida. Es una conexión causal positiva. La regla del borde negativo establece que huir de un depredador disminuye la amenaza de supervivencia. Es una conexión causal negativa. La intención de huida crece o decrece a medida que la amenaza se incrementa o disminuye. Las dos reglas definen un ciclo de retroalimentación mínima en el verbo causal del mapa cognitivo difuso (Figura 3.2.).

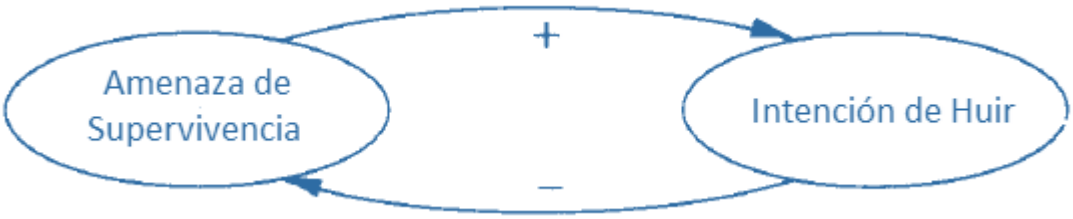


Figura 3.2. Representación del ciclo de retroalimentación entre dos nodos.

3.3 Metodología de Implementación

La presente metodología de implementación se desarrolla en base al trabajo previamente realizado por J. Conde [8], en el cual se propone el uso de un *Sistema de Comportamiento Basado en Componentes* el cual se pretende simular ambientes dinámicos, a continuación, se muestra un enfoque general del funcionamiento de la metodología.

3.3.1 Esquema de funcionamiento

La presente metodología se encuentra basada en la jerarquía de modelos usados en videojuegos y animación, en esta se consideran tres niveles de modelado:

1. Modelado de apariencia realista.
2. Modelado de movimientos realistas, suaves y flexibles.

3. Modelado de comportamientos realistas de alto nivel.

Conforme a estudios de comportamiento humano y/o animal se obtiene un modelo 3D que permita simular la forma del cuerpo del personaje en la realidad, así como sus respectivas animaciones. Posteriormente haciendo uso de los motores de física se crea un entorno adecuado para el personaje que le permita colisionar con el mundo virtual.

Los componentes requeridos para el modelado de la locomoción, el censado, el generador de intenciones y el comportamiento reactivo y cognitivo del personaje al tener un alto grado de complejidad son considerados por sí mismos como sistemas.

3.3.2 Sistema de locomoción

El movimiento de un personaje se debe modelar en función a un punto de masa o centro de gravedad, el cual se usa para definir la orientación, posición, velocidad y aceleración del personaje. Dicho centro de gravedad pose una masa determinada la cual al aplicar fuerzas especificas permite obtener dirección, una velocidad variante, e impulsos. Las fuerzas que al aplicarse permiten producir un movimiento son conocidas como fuerzas de dirección. Es necesario definir un límite sobre la cantidad de fuerza que se puede aplicar, así como la velocidad máxima que es posible alcanzar.

Las animaciones asociadas al personaje deben ser ajustadas en función del centro de gravedad del personaje, así mismo, sus animaciones deben ajustarse en base a su velocidad de animación.

3.3.3 Sistema de percepción

Para que un personaje cuente con un comportamiento realista debe ser capaz de percibir su mundo a través de una visión simulada que proporcione un campo de visión limitado, es decir, no se le permite al personaje ver más allá de un rango establecido. La percepción que posee un personaje debe permitirle provocar un mecanismo de atención con el cual el personaje pueda detectar estímulos externos y decidir el aplicar un cambio sobre su comportamiento.

Rango de percepción

El rango de percepción permite modelar y establecer las limitaciones básicas del sistema de percepción presentes en los seres vivos. Los animales en la naturaleza cuentan con habilidades de percepción que les facilitan el alimentarse. En el desarrollo del presente proyecto se hace uso del rango de visión como principal rango de percepción, con ella le será posible al pez artificial evadir colisiones cercanas a él, así como seguir su alimento si este se encuentra dentro de su rango de visión. Para fines prácticos se determinará que un objeto ha sido visto por el pez solo cuando una parte del mismo entra en su campo de visión.

3.3.4 Sistema de comportamiento

Para poder proporcionar una autonomía suficiente a los personajes es necesario que los mecanismos de comportamiento trabajen en conjunto con los mecanismos de control y de locomoción. Esto con el propósito de que el personaje cuente con patrones de comportamiento no triviales como el agruparse, vagar, buscar alimento, huir, etc. Independiente de estos es necesario implementar primero comportamientos de reflejo, los cuales le permitirán al personaje evadir obstáculos que se encuentren en su camino.

Consecutivamente a través de un mecanismo de selección de acción los comportamientos primitivos se combinan generando comportamientos motivados los cuales se activarán en función del estado propio del personaje, similar a lo mostrado en la Figura 3.2.

Comportamiento Reactivo

El encargado de proporcionar las herramientas fundamentales para reaccionar ante estímulos externos como evadir obstáculos es el sistema motor. Esta tarea no solo brinda la posibilidad de evadir colisiones con objetos estáticos del entorno, sino que también con otros personajes. J. Conde [8] menciona que “La evasión de obstáculos no sólo es un comportamiento como tal, sino más bien una fuerza continua que aleja a los personajes de las colisiones”. Dicha fuerza es generada por el terreno en que se encuentra el personaje, otros personajes posicionados junto a él, así como los límites establecidos del mundo virtual.

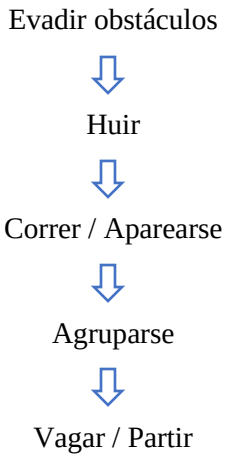
En sistemas más complejos es posible mejorar al sistema de percepción agregándole otros factores que le permitan simular un comportamiento altamente detallado, incluyendo en el

personaje hábitos, o bien información que le permita reaccionar ante la luz y temperatura del ambiente.

Comportamiento Cognitivo

El nivel cognitivo se encuentra ubicado en el nivel superior de la jerarquía de modelos usados en videojuegos y animación, gracias al cual es posible proporcionar autonomía y realismo a los personajes de videojuegos y animaciones. Para proporcionar un comportamiento realista a los personajes se hace uso de un generador de intenciones el cual funciona en base a la asignación de prioridades e intenciones que son consideradas como deseos o motivaciones.

Un personaje puede internamente contar con múltiples deseos, sin embargo, solo aquel cuya prioridad sea mayor se considerará como una intención. A continuación, se presenta un esquema que muestra los deseos básicos de un ser vivo, de acuerdo a su nivel de jerarquía.



Esta jerarquía es útil para modelar sin mucho esfuerzo un generador de intenciones el cual debe implementarse con técnicas no clásicas de lógica para que puedan obtenerse transiciones de comportamiento suaves.

3.4 Modelado Cognitivo de Peces Artificiales

Esta sección describe en detalle el desarrollo de un sistema de comportamiento cognitivo para peces y tiburones posicionados en un mundo virtual submarino. Este sistema se desarrolló en base a la metodología de implementación presentada en 3.3. basada en el trabajo de J. Conde [8].

3.4.1 Implementación

La implementación del presente trabajo se realizó mediante el motor de videojuegos Unity el cual es altamente utilizado para el desarrollo de videojuegos y animaciones, se toma como base un mundo submarino con el objetivo de brindar realismo en función a los personajes que se encontraran habitando en el (peces y tiburones), en dicho mundo se presentan como obstáculos algunos montículos submarinos los cuales en conjunto con el terreno permitirán poner a prueba las capacidades cognitivas de cada personaje.

3.4.1.1 Modelado y animación 3D

Peces Artificiales

El modelado 3D de la presa se realizó en base a un pez dorado (Figura 3.1), este es un miembro relativamente pequeño de la carpa, incluso en algunos lugares es conocido como carpa dorada o carpín. Es una de las especies de peces altamente domesticadas por lo cual es habitual encontrarlos en hogares o acuarios.

Figura 3.3. Fotografía de un Pez dorado.

El modelo 3D del pez dorado se muestra en la Figura 3.2. donde es posible apreciar la malla del modelo, así como el modelo con la textura aplicada, como es posible apreciar, la textura es la que brinda el nivel de realismo adecuado al modelo 3D, dicho modelo es un gameObject de tipo prefab diseñado en específico para Unity.

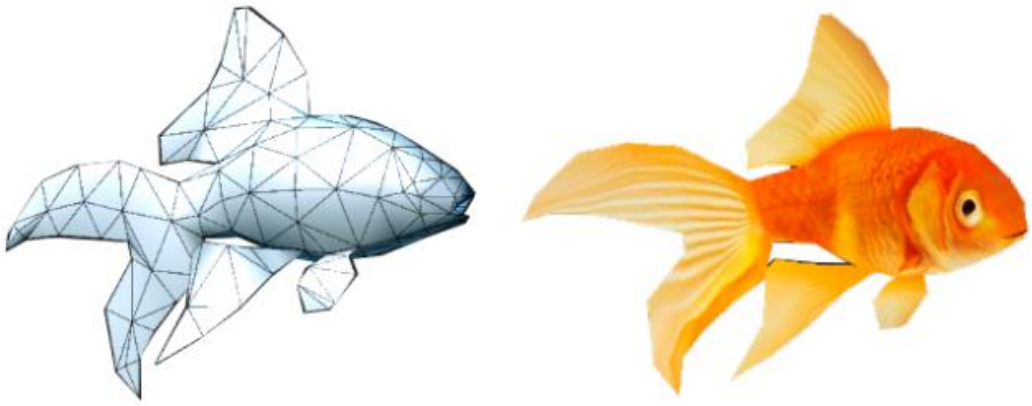


Figura 3.4. Modelo 3D del pez dorado malla (a la izquierda) y texturizado (derecha).

Así mismo el modelado 3D del depredador se realizó en base a un tiburón blanco (Figura 3.5), este es uno de los animales más famosos del mundo, en algunos lugares del mundo es conocido como jaquetón, palabra que viene de “jaque” la cual indica amenaza.

Figura 3.5. Fotografía de un tiburón blanco.

El modelo 3D del tiburón blanco se muestra en la Figura 3.6. donde es posible apreciar la malla del modelo, así como el modelo con la textura aplicada, al igual en el caso anterior, la textura es la que brinda el nivel de realismo adecuado al modelo 3D, al igual que el modelo del pez este modelo es un gameObject de tipo prefab diseñado en específico para Unity.

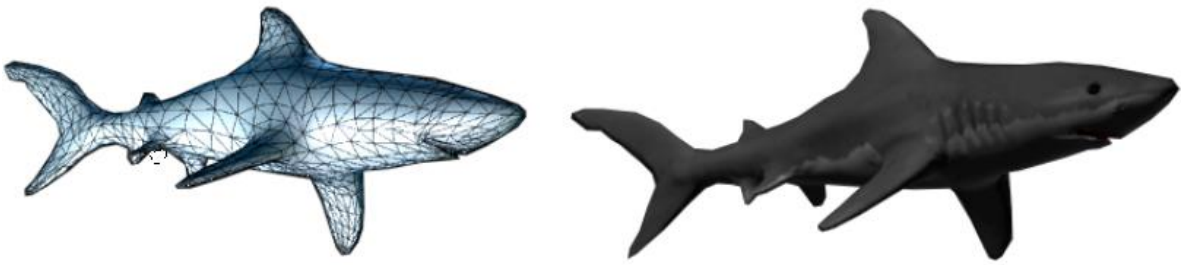


Figura 3.6. Modelo 3D del tiburón blanco malla (izquierda) y texturizado (derecha).

Respecto a la animación de nado del pez dorado se cuenta con una animación que permite simular el movimiento de nadado, con la cual mediante el componente animator de unity es posible generar otras animaciones adicionales que permitan brindar al personaje un nivel de realismo adecuado, el ciclo de animación se muestra en la Figura 3.7.

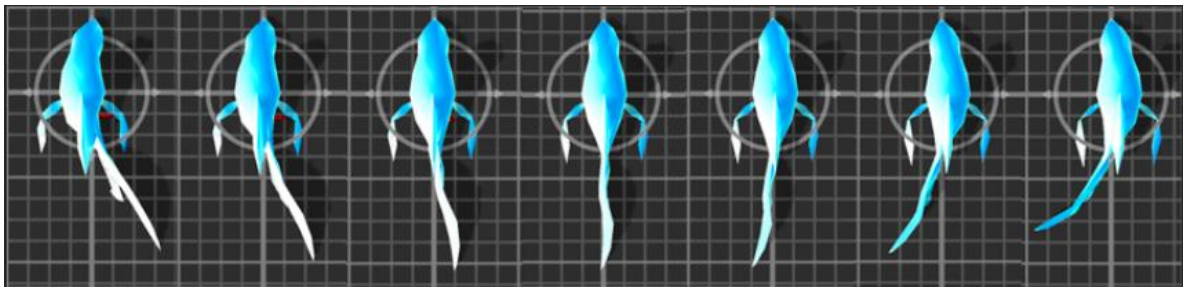


Figura 3.7. Secuencia de animación del modelo 3D del pez dorado.

Tomando en base la secuencia de animación anterior, se tomaron algunos de sus fotogramas para animar los diferentes movimientos del pez (Figura 3.8) para los movimientos en los cuales se requiere simular que el pez sube o baja, se hizo del componente de transformación propio del gameObject para rotarlo y que brinde la experiencia deseada.

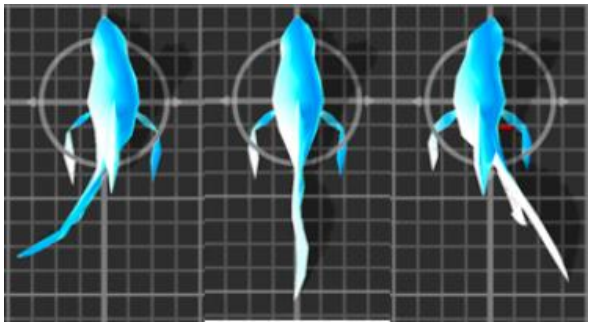


Figura 3.8. Movimientos vuelta a la izquierda, idle/frenado, vuelta a la derecha.

El tiburón al igual que el pez dorado cuenta con una animación que permite simular el movimiento de nadado, su ciclo de animación se muestra en la Figura 3.9.

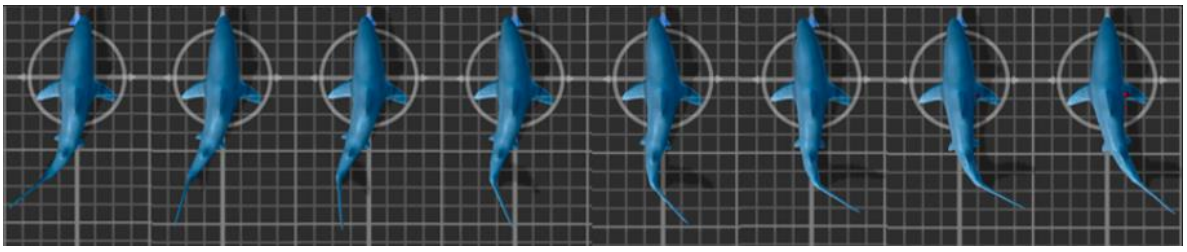


Figura 3.9. Secuencia de animación del modelo 3D del tiburón blanco.

El prefab del modelo 3D del tiburón al ser un modelo con mayor volumen incorpora sus propias animaciones para los movimientos de vuelta a la izquierda y derecha, así como del frenado.

Para que se pueda simular adecuadamente que alguno de los personajes se encuentra nadando es necesario deshabilitar la gravedad para ellos, con ello solo podrán moverse a través de la multiplicación de dos vectores asociados con la transformación del personaje, específicamente con la posición.

Cada personaje por sí mismo presenta limites mínimos y máximos respecto a su velocidad, esto ayuda a proporcionar un realismo mayor a los personajes, ya que se toma un valor aleatorio posicionado entre el rango de límites para que no todos los personajes naden a la misma velocidad.

Ambiente Submarino

El modelo virtual del ambiente debe ser capaz de proporcionar al espectador un entorno semejante a la realidad y además debe permitir poner a prueba las capacidades cognitivas y de comportamiento de los personajes.

El ambiente desarrollado se compone de un terreno que gracias a su textura se asemeja al fondo submarino, así como de formaciones rocosas las cuales brindaran la función de obstáculos estáticos.

Respecto a las dimensiones del terreno corresponden a un espacio de 100 unidades de largo por 100 unidades de ancho y su altura es de 40 unidades, con ello es posible albergar varios bancos de peces y depredadores si así se requiere, asimismo, se cuenta con un skybox acorde al entorno, se adicionaron además algunos rayos de luz que simulan la luz del sol desde la superficie, así como de algunas burbujas que nacen desde algunas partes de las formaciones rocosas (Figura 3.10).

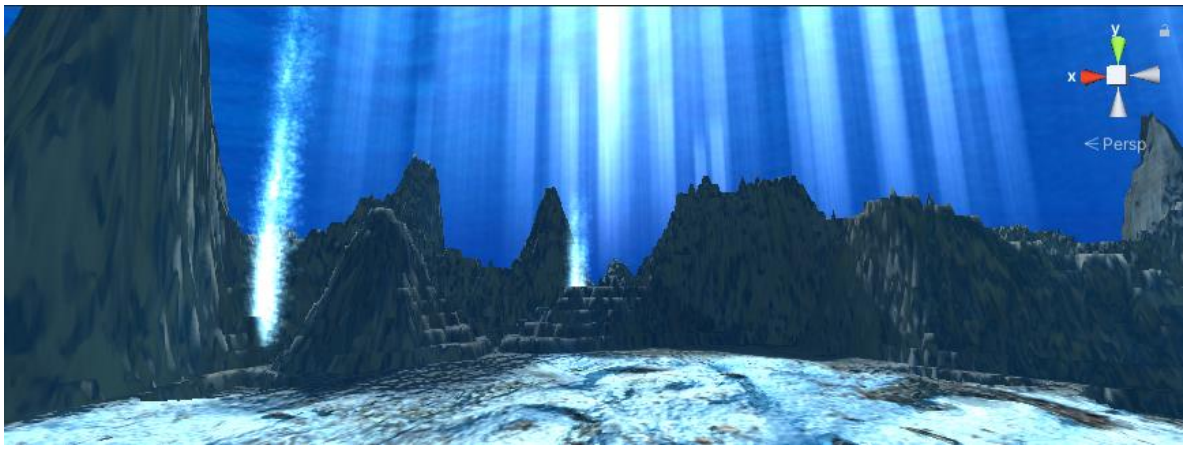


Figura 3.10. Ambiente virtual diseñado en Unity.

Adicional a las dimensiones del espacio es posible definir para cada banco de peces sus propias dimensiones estableciendo así los limites los dentro de los cuales los personajes podrán moverse.

3.4.1.2 Sistema de Percepción

Para el presente trabajo se hizo uso del sistema de percepción planteado en [8], el cual considera 4 modelos:

1. **Evadir colisión con el terreno.** Su propósito es evadir colisiones con el terreno, el cual tiene una topología particular.
2. **Evadir colisión con obstáculos fijos.** Su propósito es evadir colisiones con obstáculos fijos conocidos y obstáculos fijos desconocidos.
3. **Evadir colisión con obstáculos dinámicos.** Su propósito es evadir colisiones con objetos que estén en movimiento. En particular, con respecto a otros personajes.
4. **Contacto físico.** Su propósito es detectar el contacto físico para provocar una reacción en el personaje.

El sistema de percepción brinda al personaje la capacidad de evadir obstáculos mientras este se encuentra en movimiento, para ello se le proporciono al pez dorado un rango de visión de 270°. Conforme el pez dorado se mueve a través de su mundo virtual hace uso de una serie de raycast ubicados en su campo de visión gracias a los cuales le es posible conocer si se encuentra algún obstáculo cerca suyo.

Su funcionamiento consiste en validar primero a través de una serie de raycast cuya longitud se encuentra previamente establecida si hay algún obstáculo cerca, de ser esto verdadero se procede a evaluar por cada uno la distancia desde donde fue lanzado hasta donde golpeo el terreno y se mueve el personaje en función de la distancia más larga alcanzada, o bien se opta por tomar el camino de aquel raycast que no haya chocado contra algún objeto, este funcionamiento se ejemplifica en la Figura 3.11.

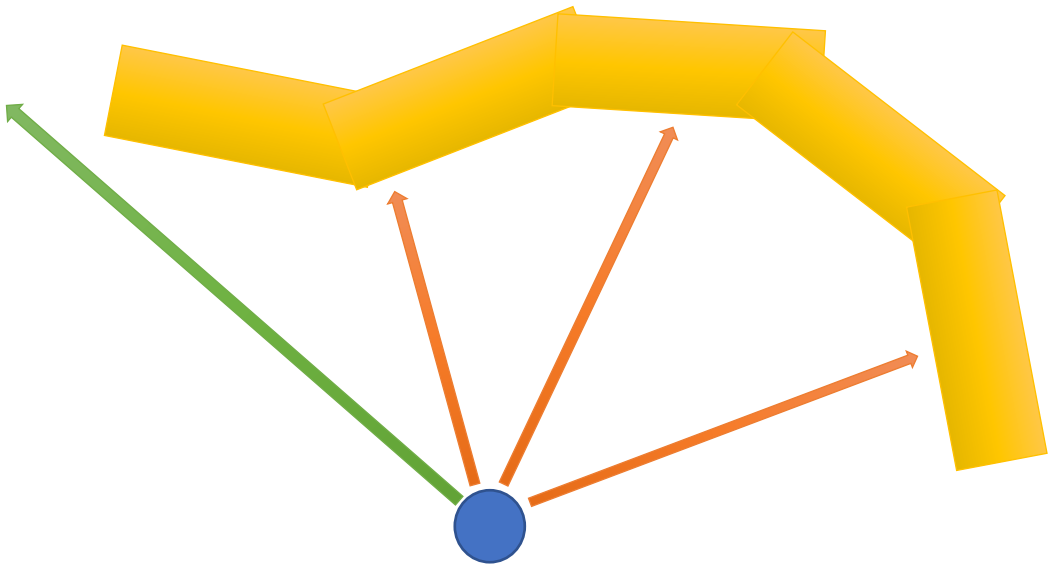


Figura 3.11. A través de los raycast se opta seguir el camino cuya distancia sea mayor.

De esta forma es posible evaluar no solamente el terreno, sino también, los obstáculos fijos. Con ello el personaje puede moverse a través de su entorno virtual, aunque no tenga conocimiento previo de él.

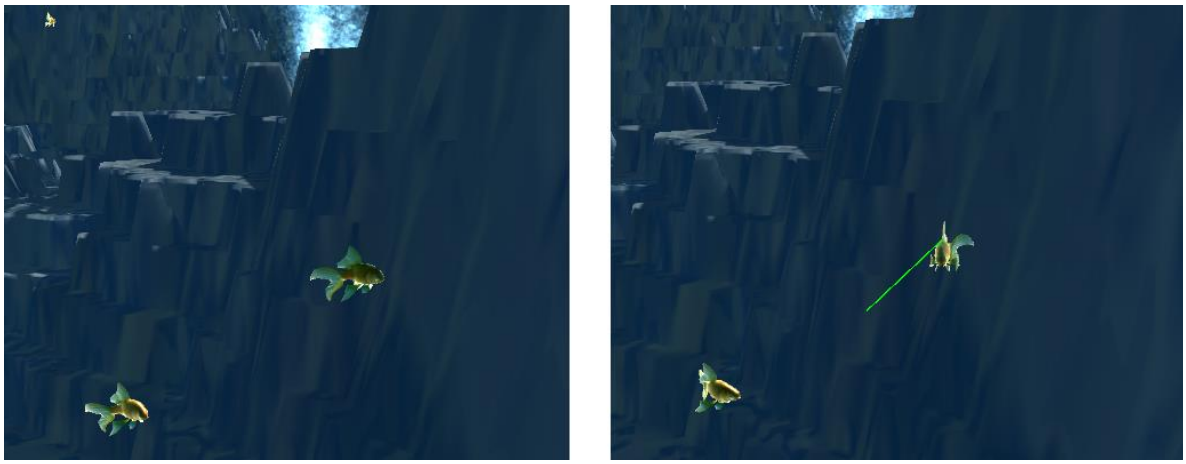


Figura 3.12. Pez dorado evadiendo una colisión contra un obstáculo estático.

Adicional al rango de visión se doto al pez dorado de un componente de tipo sphere collider (Figura 3.13), esto con el fin de emular el rango de percepción extendido presentado en [8], de esta forma cuando otro personaje entre en su rango de percepción extendido y a su vez a su campo de visión, este podrá considerarlo para poder evadirlo.

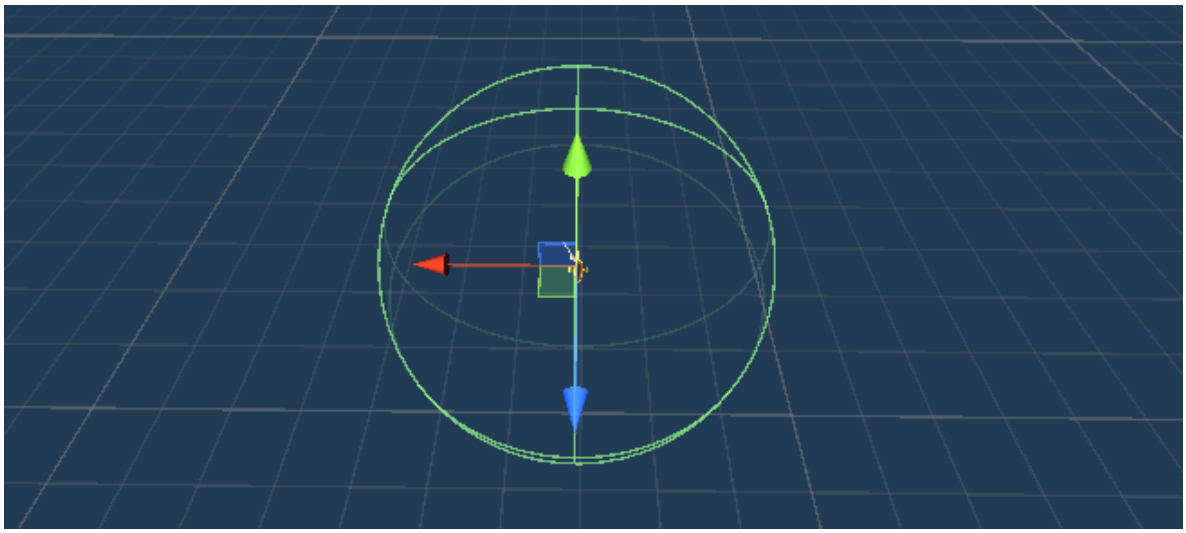


Figura 3.13.Sphere Collider que emula el rango de percepción extendido.

A su vez y para permitir que el pez dorado pueda alimentarse se hace uso del rango de percepción extendida para emular el olfato del pez, es decir, cuando la comida entre dentro de su rango de percepción el conocerá hacia dónde dirigirse para poder alimentarse, dirigiéndose en dirección de la comida que le quede más cercana, sin embargo, si no hay alimento cerca el deberá vagar hasta encontrar su alimento.

3.4.1.3 Generador de Intenciones

Para el desarrollo del generador de intenciones es necesario contar con un sistema de selección que indique la prioridad de las acciones del ser vivo, con ello y trabajando en conjunto con el sistema de percepción es posible emular comportamientos complejos de los seres vivos.

A continuación, se muestra la jerarquía de deseos de un pez, así como su FCM (Figura 3.14) ambos tomados del trabajo de J. Conde en [8] con el cual será establecer una prioridad a sus respectivas acciones:

Intención de evadir colisión

⇩

Intención de nadar rápido para alejarse del depredador

⇩

Intención de comer / Intención de descansar

⇩

Intención de agruparse

⇩

Intención de vagar / Intención de irse

De acuerdo con J. Conde [8] la red causal del Mapa Cognitivo Difuso consta de reglas o aristas que conectan a los nodos o conceptos causales dentro de una matriz de conexión. Esta matriz contiene valores en un rango de {-1, 0, 1}. La siguiente matriz describe la red causal del FCM de un pez:

	F_1	F_2	F_3	F_4	F_5	F_6	F_7
F_1	1	0	0	-1	1	0	0
F_2	1	0	1	0	-1	0	0
F_3	1	-1	0	1	-1	0	0
F_4	1	0	0	0	0	-1	0
F_5	-1	0	0	0	0	0	0
F_6	0	0	0	1	-1	0	1
F_7	1	1	0	-1	-1	-1	0

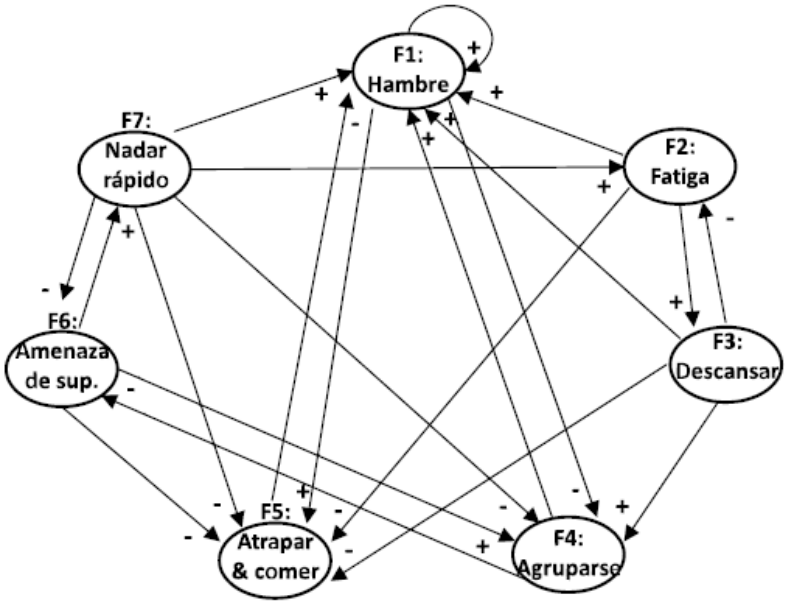


Figura 3.14. Mapa cognitivo difuso trivalente para un pez.

La información que se obtiene por parte del sistema perceptivo y del sistema motor, permite que el generador de intenciones pueda lanzar los diferentes estados internos y externos del personaje. En particular el estado mental de un pez es representado con el vector:

$$C_i(t) = [F_1\ F_2\ F_3\ F_4\ F_5\ F_6\ F_7]$$

La siguiente tabla define el cómo se trata cada uno de los conceptos causales del vector de estados mentales del pez al momento de ser lanzado.

Concepto	Intención / Deseo	Acción	Descripción	Implementación
F_1	✔		Hambre	Variable ∈ {true, false}
F_2	✔		Fatiga	Variable ∈ [0, 1]
F_3		✔	Descansar	Acción
F_4		✔	Agruparse	Acción
F_5		✔	Atrapar y comer	Acción
F_6	✔		Amenaza de supervivencia	Variable ∈ {true, false}
F_7		✔	Nadar rápido	Acción

3.4.1.4 Sistema de comportamiento cognitivo

El sistema de comportamiento cognitivo se encuentra implementado conforme a lo que se muestra en el Algoritmo 1. Pese a mostrarse pocas líneas de código el funcionamiento principal se ubica en el generador de intenciones el cual se encarga de validar las intenciones o deseos actuales del personaje y permite desencadenar las acciones pertinentes.

Algoritmo 1 Comportamiento Artificial
1. Inicializar variables internas
2. Cargar Mapa Cognitivo Difuso
3. Inicializar lista con vectores de posibles deseos del personaje
4. Inicializar variable Fatiga aleatoriamente
5. Inicializar sensor de percepción extendida.
6. **For** Cada frame **do**
7. Generador de intenciones
8. **End for**

3.4.2 Pruebas y Resultados

3.4.2.1 Pruebas

En esta sección se presentan los resultados obtenidos en la implementación del modelado cognitivo de peces artificiales, las pruebas se realizaron mediante la ejecución del proyecto en Unity:

- **Sistema de locomoción**: El sistema de locomoción se encarga de calcular y aplicar los vectores de transformación adecuados para permitir al personaje moverse a través de su entorno virtual, para poder probar el funcionamiento se asoció la rotación del personaje para que pudiese ser controlada a través del ratón y su movimiento en el espacio al teclado, mostrando un desempeño correcto, posteriormente se eliminó el uso de ratón y teclado para que el personaje pueda moverse por sí mismo sin necesidad de un agente externo.
- **Sistema de animación**: El sistema de animación funciona en conjunto con el sistema de locomoción y se encarga de reproducir las animaciones adecuadas en el

personaje en base a su estado actual, una vez implementado el sistema de percepción se validó que las animaciones que se reproducen fueran las adecuadas.

- **Sistema de percepción**: El sistema de percepción se encarga de detectar los riesgos de colisión del personaje tanto del entorno como con otros personajes, además, se encarga de identificar alguna amenaza y la ubicación cercana de alimento. Para las pruebas se ubicó al personaje en las diferentes situaciones antes mencionadas brindando un desempeño correcto.
- **Generador de intenciones**: El generador de intenciones se encarga de evaluar las condiciones internas y externas del personaje con el fin de ejecutar alguna acción, se validó mediante el envío de mensajes de alerta la acción a ejecutar y se comprobó que funcionara correctamente en los distintos escenarios:
    - Interacción Pez a Pez.
    - Interacción Pez a Tiburón.
    - Interacción Tiburón a Pez.

Figura 3.15. Sistema de percepción permitiendo evadir personajes cercanos.

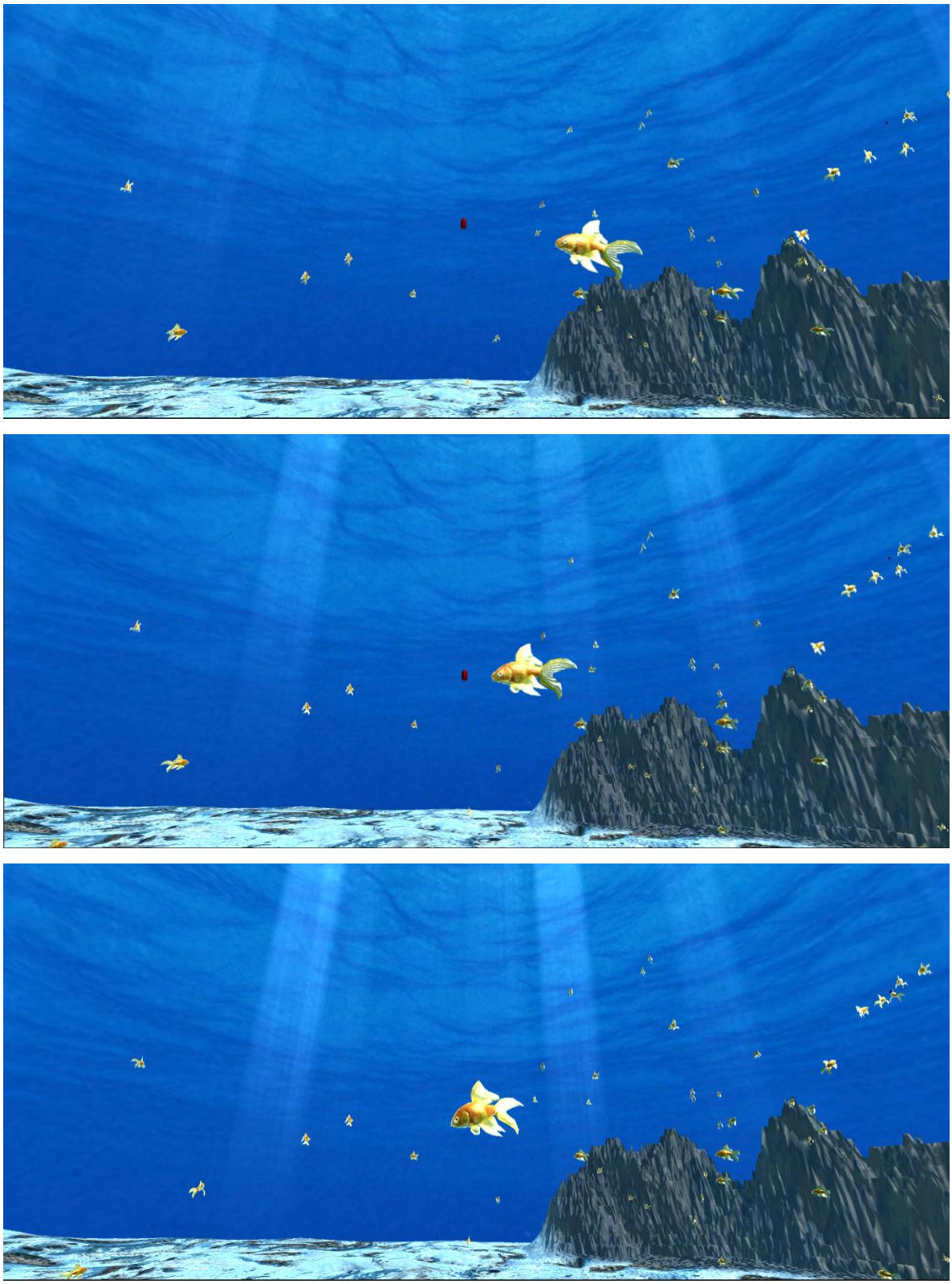


Figura 3.16. Sistema de percepción facilitando al personaje el alimentarse.

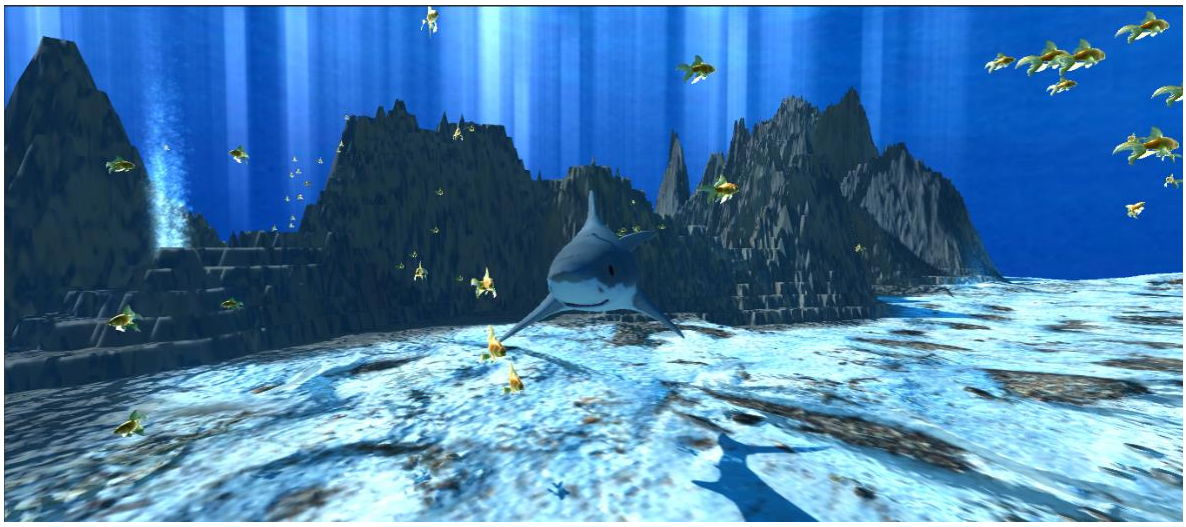


Figura 3.17. Sistema de percepción indicando al pez cuando hay un peligro cerca.

3.4.2.2 Comportamiento Resultante

El comportamiento del pez dorado se obtiene en base al mapa cognitivo difuso, para ello cuando se dispara algún deseo se multiplica el vector de deseo correspondiente por la matriz de conexión, con ello obtendremos un vector del cual se seleccionara la acción a realizar en base a las prioridades establecidas en el generador de intenciones, una vez completada la acción se multiplica nuevamente el vector previamente resultante por la matriz de conexión, esta mecánica se repite indefinidamente a lo largo de la ejecución.

En un pez los deseos que disparan un comportamiento particular son hambre, fatiga y amenaza de supervivencia. Si un pez tiene hambre el generador de intenciones se dispara el siguiente plan:

C_1	= [1 0 0 0 0 0 0]	F_1: Hambre
→ C_2	= [1 0 0 0 1 0 0]	F_1: Hambre, F_5: Atrapar y comer
→ C_3	= [0 0 0 0 1 0 0]	F_5: Atrapar y comer
→ C_4	= [0 0 0 0 0 0 0]	Vagar

Si un pez entra en estado de fatiga se dispara el siguiente plan:

C_1	= [0 1 0 0 0 0 0]	F_2: Fatiga
→ C_2	= [1 0 1 0 0 0 0]	F_1: Hambre, F_3: Descansar
→ C_3	= [1 0 0 0 0 0 0]	F_1: Hambre

→ C_4	= [1 0 0 0 1 0 0]	F_1: Hambre, F_5: Atrapar y comer
→ C_5	= [0 0 0 0 1 0 0]	F_5: Atrapar y comer
→ C_6	= [0 0 0 0 0 0 0]	Vagar

Si un pez entra en estado de alerta por una amenaza de supervivencia se dispara el siguiente plan:

C_1	= [0 0 0 0 0 1 0]	F_6: Amenaza de Supervivencia
→ C_2	= [0 0 0 1 0 0 1]	F4: Agruparse, F_7: Nadar rápido
→ C_3	= [1 1 0 0 0 0 0]	F_1: Hambre, F_2: Fatiga
→ C_4	= [1 0 1 0 0 0 0]	F_1: Hambre, F_3: Descansar
→ C_5	= [1 0 0 0 0 0 0]	F_1: Hambre
→ C_6	= [1 0 0 0 1 0 0]	F_1: Hambre, F_5: Atrapar y comer
→ C_7	= [0 0 0 0 1 0 0]	F_5: Atrapar y comer
→ C_8	= [0 0 0 0 0 0 0]	Vagar

Capítulo 4

4 Enfoque basado en la planificación de acciones orientadas a objetivos

En este capítulo, se presentan las características y funcionamiento de un sistema de comportamiento basado en la planificación de acciones orientadas en objetivos, así como los detalles de su desarrollo e implementación enfocado en un ambiente virtual el cual simula el funcionamiento de la sala de un hospital.

4.1 Planificación de acciones orientada a objetivos

La planificación de acciones orientada a objetivos (GOAP) es una arquitectura de planificación similar a STRIPS [9] diseñada específicamente para el control en tiempo real del comportamiento autónomo de los personajes en videojuegos, fue ideada por Jeff Orkin (MIT) a inicios de la década de los 2000's. Esta fue aplicada exitosamente en el videojuego F.E.A.R. (2005) para controlar el comportamiento de los personajes no jugables o NPC por sus siglas en inglés (Non Playable Character).

Si se compara la planificación de acciones orientada a objetivos (GOAP) con las máquinas de estado finito (FSM), encontramos que mientras una FSM le dice a una IA cómo comportarse en cada situación. Un sistema GOAP le dice a la IA cuáles son sus objetivos y acciones, permitiendo a la IA decidir como secuenciar las acciones para satisfacer las metas.

GOAP tiene todos los elementos de las FSM, pero trabajan de manera muy diferente (Figura 4.1), por ejemplo, las FSM definen las condiciones y comportamientos que debe seguir un NPC para el cumplimiento de alguna meta, para lograr esto las FSM pueden hacer uso de un grafo masivo o bien dividido en unidades más pequeñas. GOAP también usa grafos para su procesamiento, pero a diferencia de las FSM las acciones y objetivos están desacoplados. Las acciones son elementos libres dentro del sistema que se mezclan y combinan para cumplir los objetivos cuando están presentes. En lugar de tener una lista establecida de acciones que deben realizarse para lograr un objetivo, GOAP permite elegir entre numerosas soluciones.

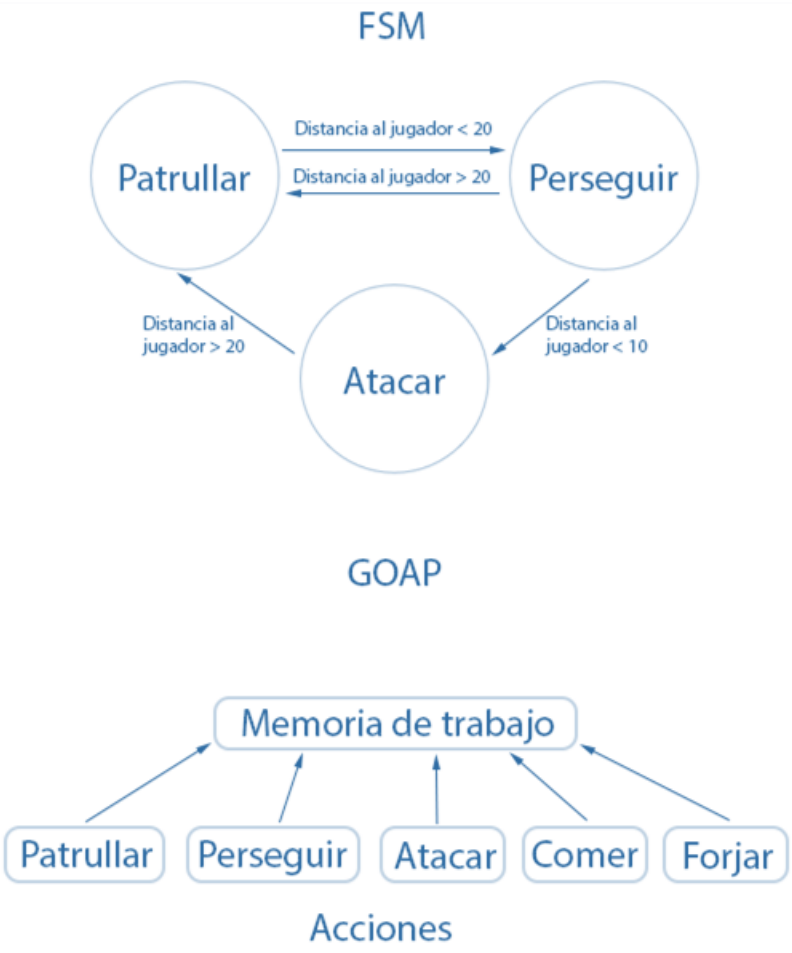


Figura 4.1. Diferencia entre un FSM y un sistema GOAP

Con ello en lugar de tener un sistema representado como un grafo dirigido, en GOAP se eliminan los enlaces, entonces, cuando se presenta una meta, se puede tomar un curso de acción apropiado seleccionado de todas las tareas disponibles en función del estado del agente y del estado del mundo, por ejemplo, si se le pidiera a un personaje que mate a un enemigo y este no tuviera un arma pasaría a elegir una pelea física, sin embargo, si tuviera un arma se presentaría un conjunto diferente de acciones elegidas (Figura 4.2) [10].

Figura 4.2. Curso de acción para matar a un enemigo sin arma y con arma.

Cada acción en GOAP tiene una precondición y un efecto (Figura 4.3), la precondición es un estado que debe cumplirse antes de que la acción pueda tener lugar, el efecto es el resultado de la acción que tiene en el estado del agente o del mundo después de que ha ocurrido. Un conjunto de acciones forma una especie de rompecabezas donde las precondiciones se pueden combinar con los efectos y los efectos coinciden con las precondiciones para crear cadenas de acción.

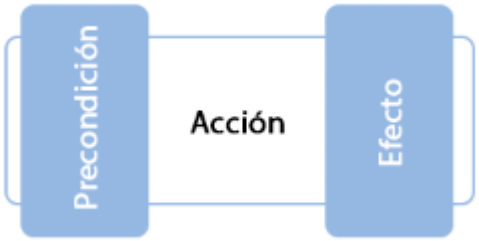


Figura 4.3. Estructura de una acción en GOAP.

Si se considera el conjunto de acciones enfocadas en comer de la figura 4.4 es posible validar que se tienen dos planes que se podrían realizar, uno para pedir pizza y otro para hornear. Un agente con estas acciones no hará nada hasta que se le presente una meta, la meta

es el estado final al cual quiere llegar el agente, no el estado inicial, en este ejemplo, se declara que la meta que se busca es igualar la variable *hambriento* a falso, si se piensa en ello, una meta recurrente de los seres vivos es evitar estar hambriento tan a menudo como sea posible.

Generalmente en GOAP los planes solo pueden ocurrir cuando los estados del mundo y/o del agente se encuentran en estado particular, en este ejemplo se consideran dos estados del mundo, *tengoIngredientes* como verdadero y *tengoNumeroPizzeria* como verdadero.

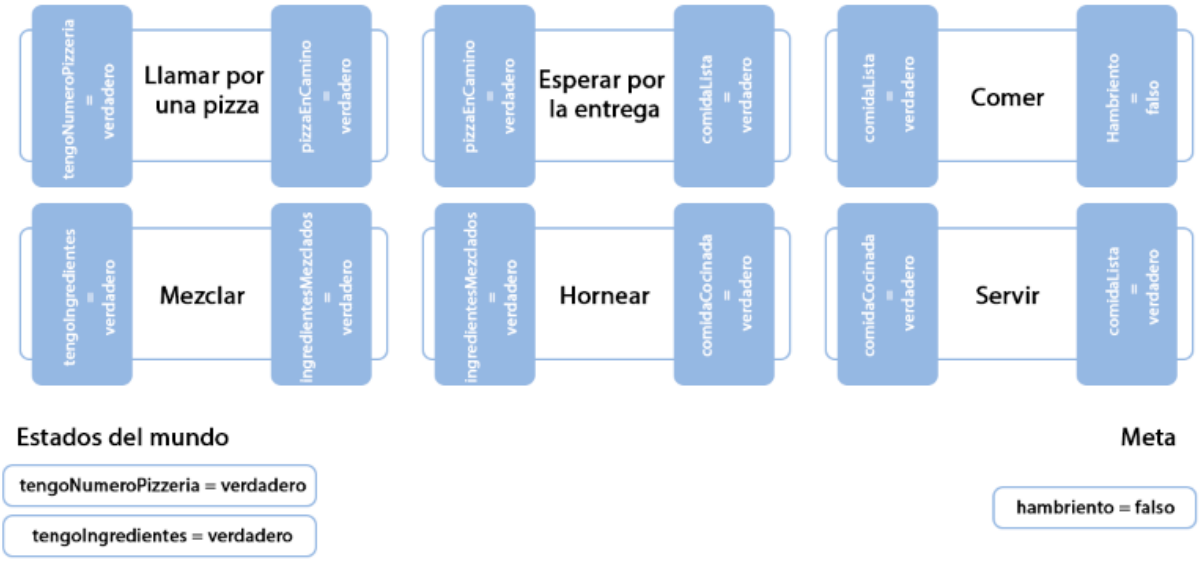


Figura 4.4. Acciones enfocadas a comer.

La etapa de planificación en GOAP siempre trabaja al revés, es decir, comienza desde la meta para validar si esta es alcanzable, esto lo hace haciendo coincidir los efectos con las condiciones previas en una cadena. Verifica todo el camino de regreso hacia los estados del mundo y/o del agente, si no puede encontrar un estado del mundo que coincida al "inicio" del plan, el plan se abandona, en el presente ejemplo contamos con dos planes posibles, ordenar una pizza o cocinar algo para comer, es posible observar que la acción de comer se ajusta en ambos planes e ilustra como las acciones se pueden mezclar y combinar para formar diferentes planes. Siempre que las condiciones previas y los efectos se ajusten, cualquier acción puede ser encadenada con otra, al tener dos planes que pueden lograr el objetivo ¿Cómo se puede seleccionar alguno? Es simple, se debe agregar un costo a los planes, para ello cada acción debe tener un costo. En particular en el ejemplo, se asignan como costos los tiempos de espera para realizar cada acción con ello es posible seleccionar el plan que genere un menor costo

(Figura 4.5). Si hay varios planes con el mismo costo entonces es posible elegir alguno de ellos al azar.

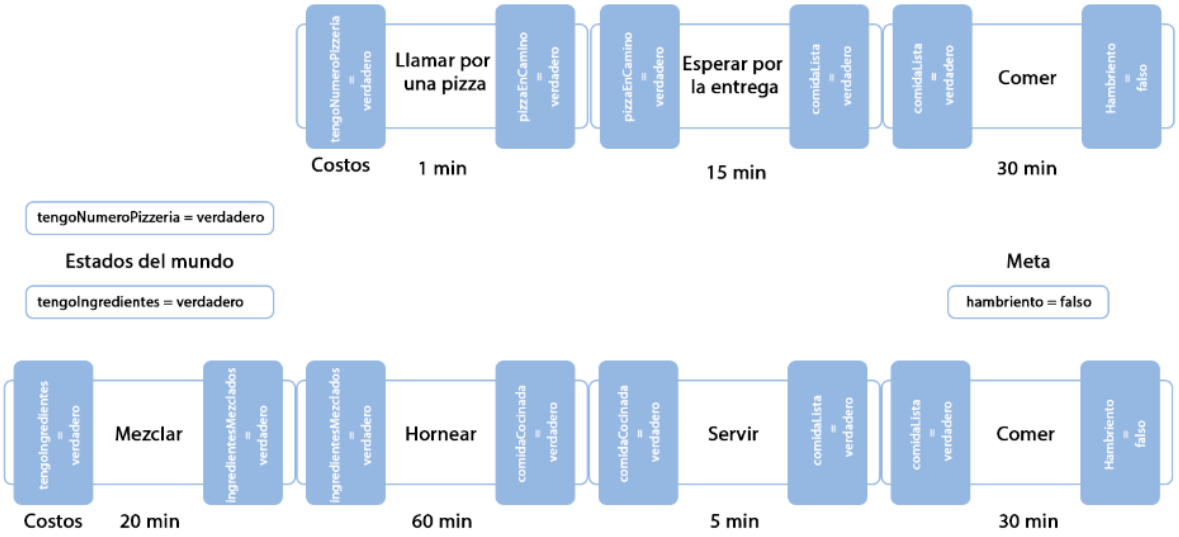


Figura 4.5. Planes generados con sus respectivos costos.

La conceptualización de un sistema GOAP se muestra en la Figura 4.6 dicha conceptualización cuenta con acciones, objetivos y estados del mundo que se introducen en un planificador de acciones, el planificador encadena las acciones de acuerdo con los objetivos y estados iniciales para determinar qué planes se pueden lograr. El planificador utiliza el algoritmo A* para encontrar el mejor plan utilizando los costos de cada acción [11]. Una vez que se haya generado un plan el agente lo ejecuta utilizando una FSM, que básicamente mueve al personaje hacia donde la acción necesita tomar lugar y entonces se ejecuta la acción. El agente trabaja a través de la cadena de acciones hasta lograr un objetivo, antes de realizar cada acción esta siempre se comprueba para ver si todavía es válida, de no serlo se descarta todo el plan y otro nuevo se genera.

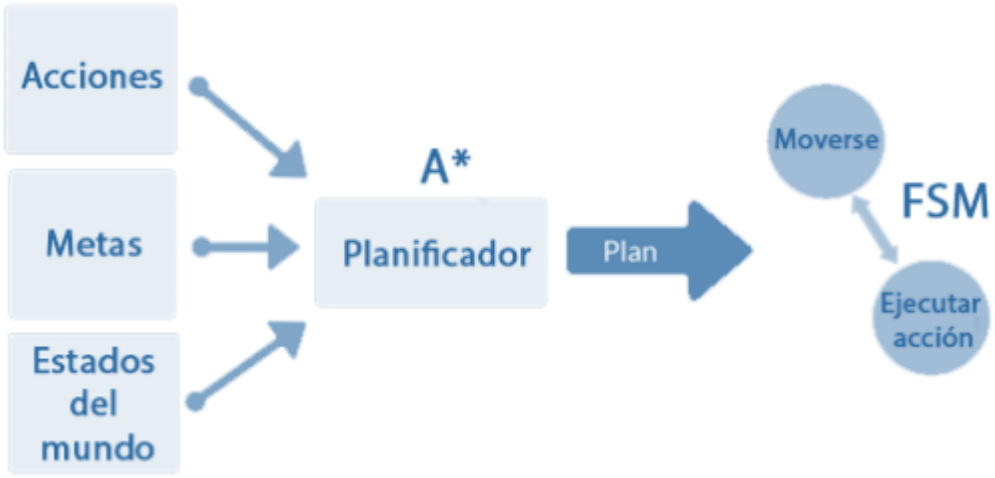


Figura 4.6. Conceptualización de un sistema GOAP.

El atractivo de GOAP radica en que es posible añadir más y más acciones al grupo disponible para el agente y estas serán seleccionadas automáticamente por el planificador [12], esto se traduce en un esfuerzo menor en programación adicional, para reconocer los grafos como en las configuraciones completas de las máquinas de estado finito los grafos son generados sobre la marcha por el planificador. Esto hace de GOAP una poderosa y flexible opción para programar los comportamientos de los NPC's.

4.2 Implementación

4.2.1 Modelado y Animación

Para el modelado de los personajes involucrados en este proyecto se hizo uso de personajes y animaciones prefabricadas desde el sitio Mixamo de Adobe, esto con el fin de proporcionar un entorno y animaciones más realistas para el usuario, sin embargo, es necesario saber administrar las animaciones del personaje en función de su comportamiento.

La animación que se reproduce sobre un personaje cambiara directamente en función del estado en el que se encuentre.

Para el personaje paciente se tienen las siguientes animaciones:

- En espera herido.

- Caminar herido.
- En espera.
- Caminar.

El cambio entre las animaciones se planifico a través de la herramienta animator de unity, dicha planificación se muestra en la Figura 4.7.

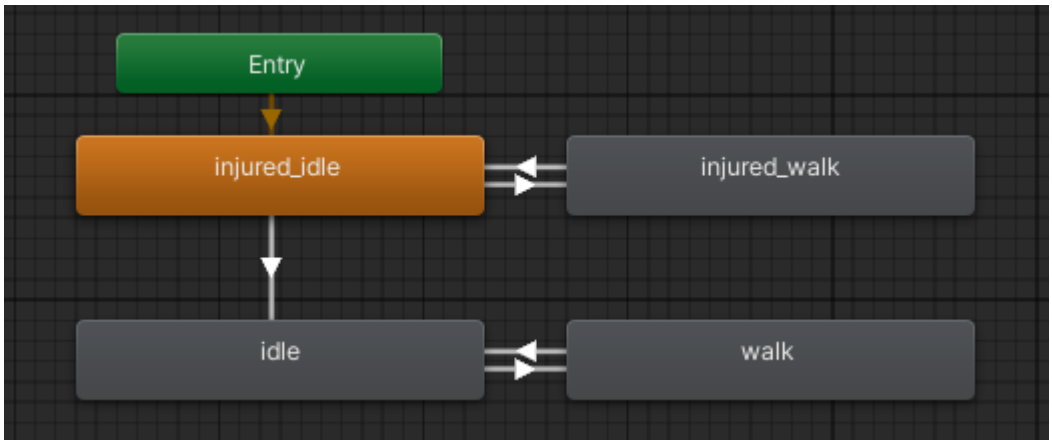


Figura 4.7. Planificación de animaciones del paciente.

Para el personaje enfermera es un poco más simple, en su caso se tienen las siguientes animaciones:

- En espera.
- Caminar.
- Tratar Paciente

El cambio entre las animaciones se muestra en la Figura 4.8.

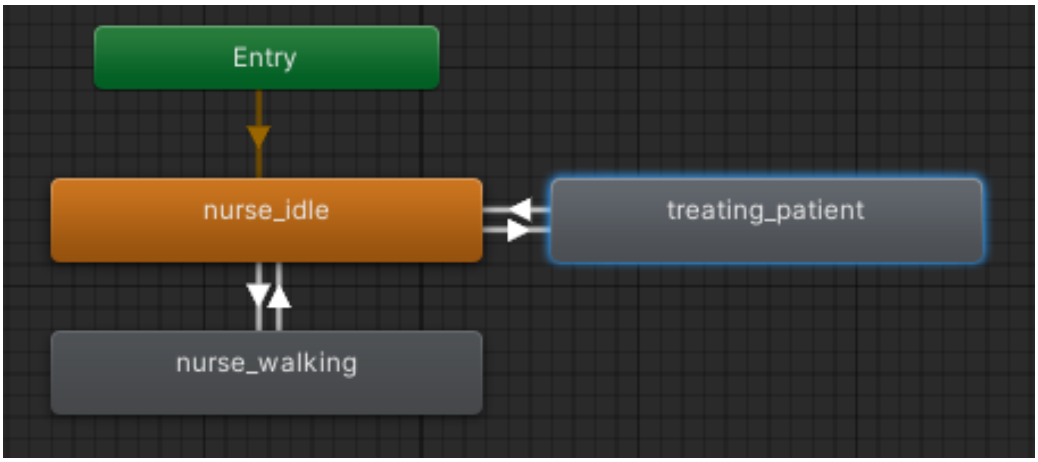


Figura 4.8. Planificación de animaciones de la enfermera.

4.2.2 Configuración de un entorno GOAP

Para poder implementar un entorno GOAP es necesario contar con un entorno virtual o escenario, en particular se creó la escena del área del área de trabajo básica de un hospital, la cual cuenta con su propia malla de navegación (NavMesh) que permite a los agentes moverse a través del escenario para cumplir con sus respectivas metas (Figura 4.9).

Figura 4.9. Malla de navegación del entorno virtual.

En este proyecto, los pacientes entran al hospital y son registrados en el escritorio ubicado en recepción, entonces se dirigen a la sala de espera, se cuenta con algunas enfermeras las cuales en caso de necesitar un descanso esperan en el área dedicada al personal, en otro caso salen de su área para atender a un paciente.

En resumen, los pacientes ingresan, se registran, pasan a la sala de espera, esperan a una enfermera, la enfermera los atiende asignándolos a un cubículo, entonces son tratados y van a casa. Con ello la enfermera se encuentra libre para atender a otro paciente o para tomar un descanso.

A lo largo de la simulación los agentes necesitan conocer donde se encuentran ubicados los waypoints en el navMesh para que puedan moverse a través de la escena. Estos se dividen de acuerdo a los dos tipos de agentes en la escena:

- **Paciente:**
  - **La puerta:** Los pacientes entran al hospital a través de la puerta, por ello un waypoint debe ser ubicado en la puerta.

    - **El escritorio:** Al entrar los pacientes deben ser registrados, por lo tanto, es necesario otro waypoint en el escritorio de recepción.
    - **Sala de espera:** Es el lugar donde los pacientes esperan a ser atendidos.
    - **Cubículos:** Es el lugar donde son asignados los pacientes para ser atendidos por las enfermeras.
    - **Fuera del mapa**: Es necesario un waypoint fuera del mapa para que los pacientes sean capases de "irse a casa" una vez que son tratados.
- **Enfermera:**
    - **Sala de descanso:** Es donde las enfermeras podrán tomar un descanso.
    - **Sala de espera:** Lugar donde los pacientes esperan a ser atendidos por las enfermeras.
    - **Cubículos:** Lugar donde las enfermeras asignan y atienden a los pacientes.

Para que los agentes puedan averiguar donde se encuentran ubicados los waypoints fue necesario añadir a cada instancia de estos una etiqueta (tag), la cual permite conocer la ubicación de los waypoints, así como emparejarlos a algún agente. El uso de etiquetas además facilita el enlazamiento de acciones, ya que todas las diferentes acciones que nuestros agentes GOAP pueden realizar deben estar enlazadas como fichas de dominó usando condiciones previas y efectos.

4.2.3 Planificación previa de las acciones del agente

Pese a que la idea de GOAP es simplista, plasmar la dinámica de la misma no lo es, debido a que hay objetivos, acciones, estados, así como otros factores se deben tomar en consideración. El sistema GOAP desarrollado funciona de la siguiente manera:

Tenemos un agente, el agente tiene una serie de objetivos, estos objetivos impulsan su secuencia de acciones, algunas acciones solo son posibles después de que se hayan completado otras acciones. Sin embargo, el entorno o el mundo virtual del juego y otros agentes también juegan un papel, por ejemplo, algunas acciones podrían no ser posibles sin la ayuda de otro agente y otras acciones pueden no ser posibles sin el uso de un recurso. Cuando se usa GOAP de hecho se está construyendo una simulación por lo que es necesario pensar en cómo se desea que luzca el juego y como los personajes van a moverse a través del entorno.

En el presente proyecto se creó una situación simple, pero con el potencial de mostrar las bondades de GOAP, la cual consiste de la simulación de un hospital con funciones básicas, en

la cual hay dos tipos de agentes: pacientes y enfermeras. Respecto a los pacientes el comportamiento deseado es que entren al hospital, se registren en recepción, vayan al cuarto de espera, sean tratados cuando una enfermera y un cubículo cuando estos estén disponibles y entonces se dirijan a casa (Figura 4.10). Cada una de estas cosas que implican mover al personaje de un lugar a otro son cada una de nuestras acciones. En GOAP cada una de estas acciones existe como una única y completa tarea que va acompañada por el agente simulando algún evento.

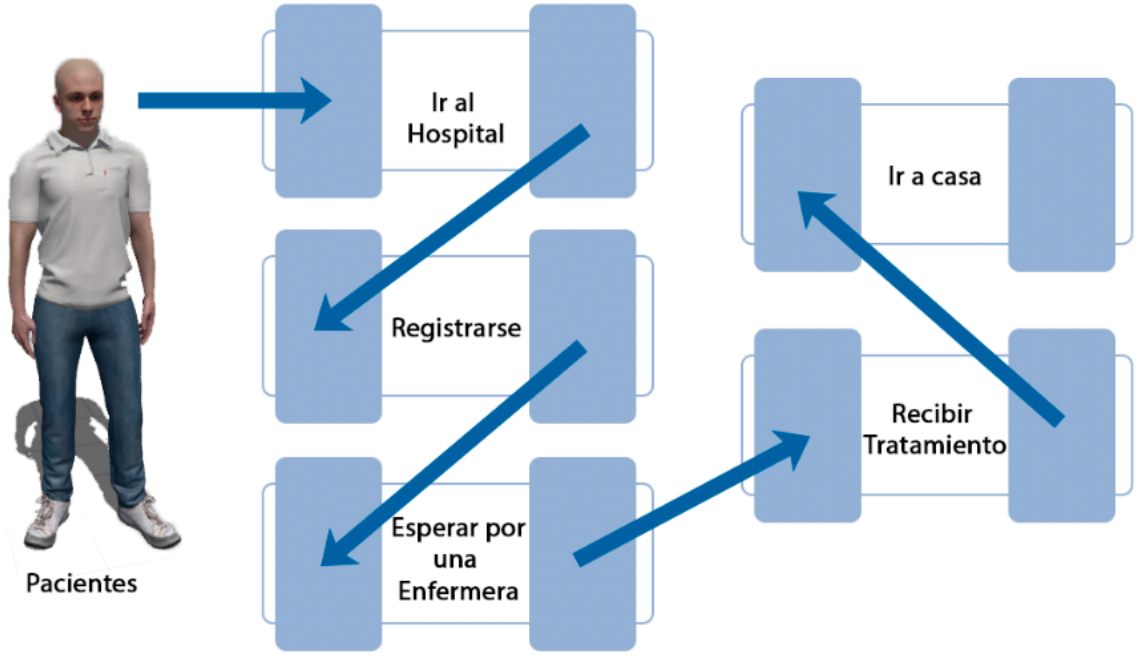


Figura 4.10. Plan de acción básico del agente.

Inicialmente se puede considerar el plan que se muestra en la Figura 4.8 si se deseará que el agente cumpliera únicamente con una dinámica. Sin embargo, es posible estructurarlo más para que el paciente tenga tres conjuntos de planes (Figura 4.11). La razón para hacer esto es porque en la simulación en si el agente tiene otros recursos y otros agentes de los cuales va a depender para continuar a través del juego actual. La importancia de GOAP radica en que cada acción se encuentra desacoplada, es decir, no se encuentra vinculada a ninguna otra, entonces, cuando se requiere crear un plan para moverse a través de las acciones de una a otra se debe buscar una meta en particular que se esté tratando de lograr y trabajar hacia atrás haciendo coincidir las acciones en conjunto para llegar a un solo plan.

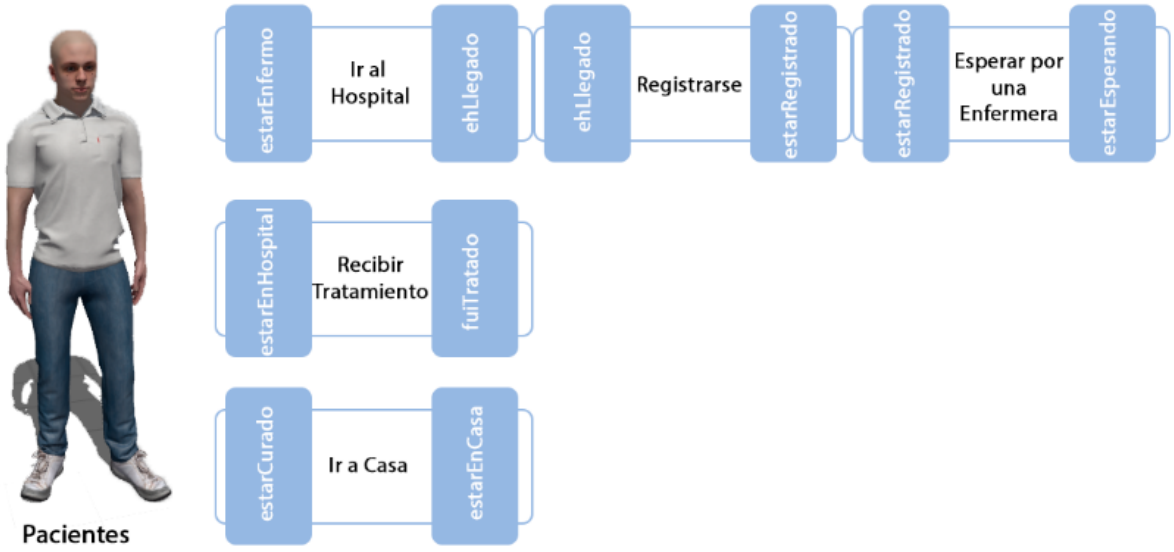


Figura 4.11. Conjunto de planes del paciente.

El primer plan consiste en ir al hospital y esperar por una enfermera, esto implicara que el paciente tenga un efecto llamado *estarEsperando*, moviéndose hacia atrás buscando la condición previa para la acción ***esperar por una enfermera*** seria *estarRegistrado*, eso significa que la acción ***registrarse*** tendrá como efecto *estar registrado* y moviéndose hacia atrás nuevamente, antes de que sea posible registrarse el paciente debe llegar al hospital (*ehLlegado)*, por ende esto conlleva que el paciente tenga que ***Ir al hospital***, de manera igual el paciente se dirige al hospital únicamente si se encuentra enfermo (*estarEnfermo)*, es necesario asegurarse de vincular suficientes acciones que tengan precondiciones y efectos que coincidan para que sea posible encontrar un plan cuando el sistema este tratando de localizar alguno.

Los otros planes que se conforman de una sola acción deben al igual contar con un efecto, el cual es el objetivo final que el agente estará buscando, por lo tanto, también es necesario asignarles una precondición.

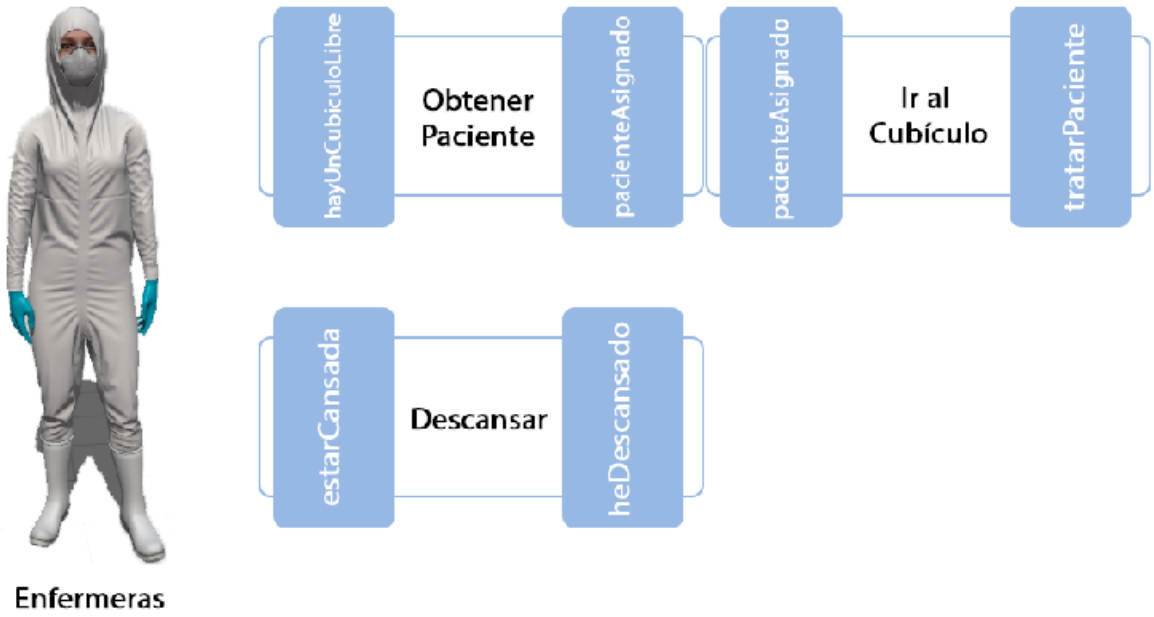


Figura 4.12. Conjunto de planes de la enfermera.

El personaje de enfermera es un poco más simplista, tiene dos conjuntos de planes para elegir (Figura 4.12), el objetivo del primer plan es tratar a un paciente, por lo tanto, se requiere de una acción que tenga como efecto *tratarPaciente* moviéndose hacia atrás es necesario una precondición para la acción ***Ir al cubículo***, dicha precondición seria tener un *pacienteAsignado*, por lo tanto, el efecto de la acción ***Obtener Paciente*** es tener un *pacienteAsignado* y su precondición seria esperar por un cubículo disponible *hayUnCubiculoLibre*, con ello será posible vincular ambas acciones permitiendo cumplir el objetivo del plan.

Para que la enfermera no se mantenga estática en un lugar cuando no hay nada que hacer se tiene un plan que consta de una acción llamada ***Descansar***, dicha acción tiene como efecto indicar que la enfermera se encuentra ha descansado (*heDescansado*), dicho plan se activará cuando la precondición que pide que la enfermera se encuentre cansada (*estarCansada*) se cumpla, por lo tanto, la enfermera se dirigirá a la sala de descanso.

El funcionamiento conceptual del sistema es el siguiente: se tiene un agente, este agente vive en un mundo virtual, dicho mundo virtual contará con un conjunto de estados del mundo los cuales proporcionaran información sobre cuantos cubículos hay disponibles, cuantos pacientes están esperando, etc., el agente podrá recurrir a esta información cuando intente determinar si puede lograr un objetivo. El agente por sí mismo contara con un conjunto de

“creencias” sobre los estados del mundo, los cuales no son en si el estado del mundo propio, además de ello, el agente también tendrá un conjunto de metas y un conjunto completo de acciones. Para alinear estas acciones y que los objetivos se puedan alcanzar es necesario el uso de un planificador, el cual se encargara de tomar la información del conjunto de los estados del mundo, estados del agente, objetivos, creencias y acciones para idear un plan, el cual será devuelto al agente el cual procederá a ejecutarlo.

4.2.4 EL planificador GOAP

Es necesario un algoritmo que permita clasificar las acciones y emparejar sus efectos con otras precondiciones para construir un plan, para ello es necesario obtener todas las acciones que un agente puede realizar, entonces dichas acciones se le proporcionan al planificador que comprueba todas ellas para validar si, en principio, alcanzables.

Posteriormente se comienza construyendo un grafo, el cual cuenta en principio con un nodo inicial, dicho nodo inicial contiene todos los estados del mundo así como sus efectos, después, se busca en todas las acciones alcanzables las que pueden realizarse dados los estados que se encuentran en el primer nodo, cada acción coincidente crea un nuevo nodo y una nueva rama en el grafo que estamos construyendo, luego, se explora cada rama para obtener otra acción que se pueda adjuntar usando las precondiciones y los efectos. Este proceso continúa en cada rama hasta que se obtenga un camino que permita satisfacer el objetivo inicial o bien cuando el planificador se quede sin acciones. Es importante tener en claro que se puede tener más de una precondición para ejecutar una acción, lo cual en muchas ocasiones tiene que ver con los estados del mundo.

Si se encuentra un camino, este camino se convierte en la cola de acciones del agente y luego el agente comienza a realizar dichas acciones.

En teoría se tiene un grafo, mediante el cual el planificador se encargará de vincular las acciones juntas en un plan, pero en la práctica, en lugar de vincular las acciones unas a otras se crea un conjunto de nodos que están vinculados entre sí, y luego simplemente el planificador se encarga de mantener o señalar una acción en particular

4.3 Pruebas y Resultados

4.3.1.1 Pruebas

En esta sección se presentan los resultados obtenidos de la implementación del enfoque basado en objetivos de una sala de un hospital, las pruebas se realizaron mediante la ejecución del proyecto en Unity:

- **Sistema de locomoción**: El sistema de locomoción se encarga de calcular y aplicar los vectores de transformación adecuados para permitir al personaje moverse a través de su entorno virtual, para poder probar el funcionamiento al momento de la ejecución se validó que se desplazara correctamente a través de su malla de navegación.
- **Sistema de animación**: El sistema de animación funciona en conjunto con los estados internos del agente y se encarga de reproducir las animaciones adecuadas en el personaje en base a su estado actual, una vez implementado el sistema se validó que las animaciones que se reproducen fueran las adecuadas en función del estado del personaje.
- **Planificador GOAP**: Se valido que el planificador se encargara de calcular los planes, así como a proceder a asignar las acciones a realizar a los personajes, nuevamente el funcionamiento fue el adecuado.

4.3.1.2 Resultados

El comportamiento de los personajes obtenido fue el correcto de acuerdo a la planificación de sus acciones mostradas en las Figura 4.11. y 4.12

A continuación, se muestra una tabla con las precondiciones, acciones y efectos obtenidos por parte de las enfermeras:

Precondiciones		**Acción**	**Efectos**	
Estados del mundo	**Creencias**		**Estados del mundo**	**Creencias**

pacienteEnHospital		Obtener Paciente	cubiculoOcupado	pacienteAsignado
cubiculoOcupado	pacienteAsignado	Ir al Cubículo	cubiculoLibre	tratarPaciente
	estarCansada	Descansar		heDescansado

Figura 4.13. Enfermera en dirección a asignar un paciente.

Figura 4.14. Una vez asignado un paciente se dirige al cubículo para atenderlo.

A continuación, se muestra una tabla con las precondiciones, acciones y efectos obtenidos por parte de los pacientes:

Precondiciones		**Acción**	**Efectos**	
Estados del mundo	**Creencias**		**Estados del mundo**	**Creencias**
	estarEnfermo	Ir al Hospital		ehLlegado
	ehLlegado	Registrarse		estarRegistrado
	estarRegistrado	Esperar por una Enfermera	pacienteEnHospital	estarEsperando
pacienteEnHospital		Recibir Tratamiento		fuiTratado
	fuiTratado	Ir a casa		estarEnCasa

Figura 4.15. Al llegar el paciente este procede a registrarse.

Figura 4.16. En la sala de espera se le asigna un cubículo de atención.

Figura 4.17. Una vez es atendido se dirige a casa.

Capítulo 5

5 Conclusiones y trabajo futuro

El objetivo principal de la elaboración de esta tesis fue proporcionar una propuesta para el modelado cognitivo en el campo de la animación y videojuegos para ello se aprovecharon las bondades que proporciona el motor de videojuegos Unity.

En este documento se muestra el desarrollo de una propuesta previamente realizada por J. Conde en [8], acorde a ello, se realiza una segunda propuesta.

En cuanto a desarrollo si se compara una propuesta con otra podría concluir que el desarrollo basado en objetivos presenta una mayor facilidad respecto al diseño de los planes de un personaje, así como de la creación de nuevas tareas gracias a la modularidad de las mismas, sin embargo, en cuanto al planificador de acciones su desarrollo es más complejo, además de ello, al tener acciones con precondiciones que dependen de los estados del mundo se puede llegar a obtener un comportamiento un tanto irreal, ya que los personajes no usan sus "sentidos", en su lugar, hacen uso de sus creencias o estados internos en conjunto con los estados del mundo para evaluar si se puede ejecutar una acción, por ello, este modelo de desarrollo me parece más apropiado para videojuegos.

Respecto al modelado basado en mapas cognitivos difusos, puedo concluir que su diseño es en si un gran reto, ya que es necesario combinar las intenciones y/o deseos junto con sus

respectivas acciones para poder mapear posteriormente el mapa a una matriz, adicional a ello cuando se realiza la multiplicación de intenciones con la matriz de conexiones pueden llegar a presentarse casos donde en ningún plan se ejecuta una acción que se había contemplado inicialmente, sin embargo, en este enfoque se puede evitar que los personajes conozcan previamente los estados del mundo, ya que ellos pueden explorar su entorno e ir descubriendo e interactuando con él, dada su complejidad en su diseño considero que este es mas adecuado para el campo de la animación.

Aun es posible encontrar un gran campo sin explorar en el campo de la animación y videojuegos, por lo que este trabajo se podría complementar con algunas otras técnicas de planeación como STRIPS por ejemplo.

6 Bibliografía

[1] Chavarría Sánchez, L. (2017). Computación cognitiva. Recuperado el 20 de agosto del 2021, de RepositorioTEC Sitio web: https://revistas.tec.ac.cr/index.php/investiga_tec/article/view/3026/2774

[2] Funge, J. (2018). Al for Games and Animation. Boca Raton, FL: CRC Press.

[3] Osorio Pastrana, A., Gómez, A., & Caro Piñeres, M. (2017). Modelado cognitivo en educación. Acta ScientiÆ InformaticÆ, 1(1). Recuperado a partir de https://revistas.unicordoba.edu.co/index.php/asinf/article/view/1168

[4] Iglesias, A. (2013). Modelo computacional cognitivo de toma de decisiones basado en el conocimiento: aplicación en la inferencia de explicaciones. Tesis doctoral. Universidad Complutense de Madrid.

[5] Funge, J. (2000). Cognitive modeling for games and animation. Communications of the ACM Volume 43, Issue 7. July 2000, pp. 40–48. DOI:https://doi.org/10.1145/341852.341862

[6] Cossu, S. (2021). Beginning Game AI with Unity. London, Uk: Apress.

[7] Dickerson, Julie & Kosko, Bart. (1994). Virtual Worlds as Fuzzy Cognitive Maps. Presence. 3. 173-189. 10.1162/pres.1994.3.2.173.

[8] Conde, J. (2013). Modelado cognitivo en videojuegos. Tesis de maestría. Benemérita Universidad Autónoma de Puebla.

[9] Orkin, J. (2006, March). Three states and a plan: the AI of FEAR. In Game developers conference (Vol. 2006, p. 4).

[10] Orkin, J. (2003). Applying goal-oriented action planning to games. AI game programming wisdom, 2, 217-228.

[11] Orkin, J. (2005, June). Agent Architecture Considerations for Real-Time Planning in Games. In AIIDE (pp. 105-110).

[12] Orkin, J. (2004, July). Symbolic representation of game world state: Toward real-time planning in games. In Proceedings of the AAAI Workshop on Challenges in Game Artificial Intelligence (Vol. 5, pp. 26-30).